



debian

Debian-Pakete bauen
mit
Git-Buildpackage

Dipl.-Ing. Mechtilde Stehmann

und

Dr. Michael Stehmann

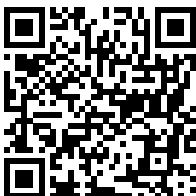
4. Mai 2025

Deutsche Version:



<https://ddp-team.pages.debian.net/dpb/BuildWithGBP.pdf>

Englische Version:



https://ddp-team.pages.debian.net/dpb/en_US/BuildWithGBP.pdf

Inhalt

I. Überblick	1
1. Lizenz	3
2. Zum Titel des Buches	5
3. Wer sollte dieses Buch lesen?	7
4. Wie entsteht dieses Buch?	9
4.1. Motivation	9
4.2. Baustellen	10
4.3. Werkzeuge	10
5. Konventionen	13
5.1. System	13
5.2. Terminologie	13
5.3. Typographie	13
5.4. Darstellung des Quellcodes	13
6. Kurzanleitung	15
6.1. Vorbereitung der Build-Umgebung	15
6.2. Nutzung des Programmskriptes	16
II. Grundlagen	17
7. Lektüre	21
7.1. Debian Free Software Guidelines	21
7.2. Debian Policy-Handbuch	21
7.3. Debian Entwicklerreferenz	21
7.4. Referenz für <i>Git-Buildpackage</i>	22
7.5. Handbuch für Debian-Betreuer	22
7.6. Debian Leitfaden für Neue Paketbetreuer	22
7.7. Weitere Informationen	22
8. Was ist ein Debian-Paket?	23
9. Auswahl der zu paketierenden Software	25
10. Prüfung der Quellen	27
10.1. Lizenzprüfung	27
10.1.1. <i>debmake</i>	27
10.1.2. <i>licensecheck</i>	27
10.1.3. <i>scan-copyrights</i>	28
10.1.4. <i>licensing</i>	28
10.1.5. <i>cme</i>	28

10.1.6.	Manuell	29
10.1.7.	Stolperfallen	29
10.2.	Feststellung der Programmiersprache	29
10.3.	Prüfung der Abhängigkeiten	29
10.3.1.	Abhängigkeiten ermitteln mit <i>packages.debian.org</i>	30
10.3.2.	Abhängigkeiten ermitteln auf <i>codesearch.debian.net</i>	30
10.3.3.	Suche nach Abhängigkeiten auf der Konsole	30
10.4.	Änderungen am Quellcode	30
10.4.1.	Ganze Dateien ausschließen	30
10.4.1.1.	Auflistung der auszuschließenden Dateien	31
10.4.1.2.	Fallunterscheidungen	31
10.4.1.3.	Benennung der Pakete beim Ausschluss von Dateien	32
10.4.2.	Änderungen in einzelnen Quellcode-Dateien (Patches)	32
10.4.2.1.	Patches mit <i>quilt</i>	32
10.4.2.2.	Patches in einem <i>Patch-Queue</i> -Zweig	33
11.	Versionierung der Pakete	35
11.1.	Paketname	35
11.2.	Versionierungsschema	35
11.3.	<i>apt</i> und <i>dpkg</i>	36
11.4.	<i>uscan</i> und die Datei <i>debian/watch</i>	36
12.	<i>dh_make</i>	37
13.	Java-Pakete bauen	39
13.1.	Herausforderungen	39
13.2.	Anwendungen und Bibliotheken	39
13.2.1.	Java-Programme paketieren	40
13.2.2.	Java-Bibliotheken paketieren	40
13.2.3.	Name des Java-Paketes	40
13.3.	Abhängigkeiten bei Java-Paketen	40
13.3.1.	Weitere Abhängigkeiten feststellen	40
13.3.2.	Abhängigkeiten ermitteln	41
13.4.	Build-Systeme für Java-Pakete	41
13.4.1.	Das Build-System <i>maven</i>	41
13.4.2.	Paketieren mit <i>maven</i>	42
13.4.3.	Paketieren mit <i>ant</i>	45
13.4.4.	Paketieren mit <i>gradle</i>	45
13.5.	Java-Pakete bauen ohne Build-System	45
14.	Mozilla-Erweiterungen bauen	47
14.1.	Quellen der Erweiterungen	47
14.2.	Integration ins Dateisystem	48
15.	Python-Pakete bauen	49
16.	Dokumentation paketieren	51
16.1.	Dokumentation für Debian bauen	51
16.2.	Upstream-Dokumentation bauen	51
17.	Metapakete bauen	53
17.1.	Kein Upstream-Quellcode	53

17.2.	Natives Debian-Paket	53
17.2.1.	<i>debian/source/format</i>	53
17.2.2.	<i>debian/control</i>	53
17.2.3.	<i>debian/rules</i>	53
17.2.4.	<i>debian/changelog</i>	53
18.	Konfiguration zur Installation	55
18.1.	<i>debconf</i>	55
18.2.	<i>debconf-common</i>	55
19.	System einrichten	57
19.1.	Abhängigkeiten für das Programmskript	57
19.1.1.	Generelle Abhängigkeiten	57
19.1.2.	Abhängigkeiten für das Bauen von Java-Paketen	58
19.1.3.	Abhängigkeiten für Erweiterungen, die <i>Zip</i> -Archive sind	58
19.2.	Verzeichnisse und Dateien	58
19.2.1.	Pfade zu den Projekten	58
19.2.2.	Konfigurationsdateien	59
19.2.2.1.	Für jedes Projekt	59
19.2.2.2.	Für viele Projekte	60
19.2.2.3.	Fingerprint des Maintainer-Schlüssels	60
19.2.3.	<i>.bashrc</i>	60
19.3.	<i>PBuilder</i> einrichten	60
19.3.1.	<i>Chroot</i>	60
19.3.2.	Konfiguration des <i>Pbuilders</i>	61
19.3.3.	Hooks einrichten	64
19.3.4.	Hooks - Beispiele	65
19.3.4.1.	Hook A	65
19.3.4.2.	Hook B	65
19.3.4.3.	Hook C	66
19.3.4.4.	Hook D	66
19.3.4.5.	Hook E	66
19.3.4.6.	Hook F	66
19.3.4.7.	Hook G	67
19.3.4.8.	Hook H	67
19.3.4.9.	Hook I	67
19.3.5.	Alternative <i>Chroot</i> -Umgebungen	67
19.4.	Konfiguration von <i>sbuid</i>	68
19.5.	Weitere <i>Chroot</i> -Systeme	69
19.6.	<i>Quilt</i> fürs Patchen einrichten	69
20.	Git einrichten	71
20.1.	Branches	71
20.2.	Mergen	71
20.3.	<i>gbp.conf</i>	71
20.3.1.	Reihenfolge	72
20.3.2.	Abschnitte in der <i>gbp.conf</i>	72
20.3.3.	Syntax der Optionen	72
20.3.4.	Beispiel	72
20.4.	Git-Repositoryen auf eigener Infrastruktur	74
20.4.1.	Lokales Git-Repository	74
20.4.2.	Eigener Git-Server	74

21. Salsa-Repositorien	75
21.1. Salsa-Account anlegen	75
21.2. Anlage eines Salsa-Repositoriums	75
21.3. Salsa-Repositorium für das Java-Team	76
21.3.1. Quelle des Skripts	76
21.3.2. Abhängigkeiten	76
21.3.3. Zugangstoken beschaffen	76
21.3.4. Token eintragen	77
21.3.5. Skript aufrufen	77
21.4. Aufgaben auf <i>salsa.debian.org</i>	78
21.4.1. Merge Request	78
22. Paketieren jenseits vom Zweig Unstable	79
22.1. Security-Updates	80
22.2. (Old-)Stable-Proposal	80
22.2.1. Fehlerbericht	81
22.2.2. Anforderungen an einen Patch	82
22.2.3. Abhängigkeiten zu Mozilla-Paketen	82
22.3. Stable-Backports	82
22.4. Backports-Repositorium	82
22.5. Experimental	83
22.6. Versionierung	83
23. Zum Start eine E-Mail	85
23.1. ITP - Intent To Package	85
23.2. RFP - Request For Package	86
23.3. ITA - Intent To Adoption	86
23.4. RFA - Request for Adoption	87
23.5. RFH - Request For Help	87
23.6. O - Orphaned	87
23.7. RFS - Request For Sponsor	87
24. Änderungen am Fehlerbericht	89
24.1. Anpassung des Schweregrades	89
24.2. Anpassung des Titels	89
24.3. Änderung des Maintainers	89
24.4. Öffnen eines geschlossenen Fehlerberichtes	89
24.5. Schließen eines Fehlerberichtes	90
24.6. <i>usertags</i> hinzufügen	90
25. Reportbug einrichten	91
26. Schwierigkeiten überwinden	93
26.1. Ein Paket loseisen	93
26.1.1. Beantragung einer Entsperrung	93
26.2. Releasekritische Fehler beheben	94
26.3. Paket aus Repositorien entfernen	94
26.3.1. Feststellung der Rückwärtsabhängigkeit	94
26.3.2. Fehlerbericht	94
27. Autopkgtest	95
28. Lintian-Meldungen	97

29. Reproduzierbare Builds	99
29.1. reprotest	99
30. piuparts	101
III. Wie ein Shell-Skript hilft, ein Debian-Paket zu bauen	103
31. Erste Schritte im Programmskript	105
31.1. Der Anfang steht am Schluss	106
31.2. Und das sieht der Nutzer als Erstes	106
31.3. Projektname abfragen	107
31.4. Weiterer Ablauf	110
32. Anlegen eines neuen Projektes	111
32.1. Konfigurationsdatei erstellen	111
32.1.1. Abfrage allgemeiner Variablen für die Konfigurationsdatei . .	113
32.1.2. Abfrage spezieller Variablen für die Konfigurationsdatei . . .	122
32.1.2.1. Ermittlung der Plugin-Pfade	122
32.1.2.2. Variablenabfrage für Java-Pakete	123
32.1.2.3. Variablenabfrage für Mozilla-Erweiterungen	125
32.1.2.4. Variablenabfrage für Python3-Pakete	126
32.1.3. Speichern der Konfiguration	127
32.1.4. Beispiel einer Konfigurationsdatei	129
32.2. Anlegen der Infrastruktur	129
32.2.1. Anlegen der notwendigen Verzeichnisse	129
32.2.2. Definition der Pfade	130
32.2.3. Anlegen der Log-Datei	130
32.3. Git-Repositoryen	131
32.3.1. Gibt es bereits ein Git-Repositoryum?	131
32.3.2. Auswahldialog	132
32.4. Neuanlage eines lokalen Git-Repositoryums	135
32.4.1. Git-Repositoryum anlegen	135
32.4.2. Name und E-Mail-Adresse ins Git-Repositoryum einfügen . .	136
32.4.3. Repositoryum auf <i>salsa.debian.org</i>	148
32.4.3.1. Manuell	148
32.4.3.2. Innerhalb des Java-Teams	149
32.4.4. Remoteserver anzeigen	149
32.5. Klonen von <i>salsa.debian.org</i>	150
32.5.1. Bestimmung der Git-Zweige	151
32.5.2. Git-Zweige ermitteln	154
32.5.3. Git-Zweig Distribution zuordnen	154
32.5.4. Name und E-Mail-Adresse hinzufügen	155
32.6. Import eines Debian-Quellcode-Paketes	156
32.7. Import für das Sponsoring	158
32.8. GnuPG-Schlüssel verfügbar?	160
32.9. Fingerprint nutzen	161
32.10. Start des Paketierens	164
33. Arbeiten in einem angelegten Projekt	165
33.1. Konfigurationsdatei laden und editieren	165
33.2. Ändern von Zeilen in der Konfigurationsdatei	167

33.3.	Zeile in Konfigurationsdatei einfügen	168
33.4.	Auswahl eines Git -Zweiges	169
33.4.1.	Prüfung mit <i>git status</i>	170
33.4.2.	Fehlermeldung und -behebung	171
33.4.3.	Auswahl der Debian -Branches	172
33.4.4.	Dialog zur Auswahl eines Branches	173
33.4.5.	Eintrag ändern	174
33.4.6.	Konfiguration einlesen	175
33.4.7.	Kein oder nur ein Branch existiert	176
33.5.	Aufgabenauswahl	177
34.	Bauen einer neuen Version	181
34.1.	Änderungen von <i>Salsa</i> herunterladen	181
34.2.	Werkzeuge zum Herunterladen der Upstream-Sourcen	183
34.3.	Herunterladen auf klassische Weise	186
34.3.1.	Archiv-Formate	186
34.3.2.	Herunterladen des Quellcodes	186
34.3.2.1.	Herunterladen	188
34.3.2.2.	Kopieren des Quellarchivs	190
34.3.3.	Komprimierung ermitteln	192
34.3.4.	Upstream-Version ermitteln	193
34.3.5.	Dateien aus Upstream-Archiv ausschließen	197
34.3.6.	Debian -Quellcode-Datei erzeugen	205
34.3.7.	Signatur prüfen	208
34.3.7.1.	Signatur-Datei herunterladen	208
34.3.7.2.	Prüfung der Signatur	209
34.3.8.	Link durch Kopie ersetzen	209
34.3.9.	<i>gbp</i> -Konfigurationsdatei	210
34.3.10.	Prüfung des Git -Repositoriums	216
34.3.11.	Import nach Git	218
34.4.	Herunterladen und Importieren mit <i>uscan</i>	221
35.	Bauen einer neuen Revision	225
35.1.	Anlegen des Debian -Verzeichnisses	226
35.2.	Abfrage: Bauen mit <i>mh-make</i> ?	227
35.3.	Sollen die Debian -Dateien angezeigt werden?	228
35.4.	Dateien im Verzeichnis <i>debian/</i>	229
35.4.1.	Anzeigen der Debian -Dateien	229
35.4.2.	<i>debian/source/format</i>	231
35.4.3.	<i>debian/source/include.binaries</i>	233
35.4.4.	<i>debian/upstream/metadata</i>	233
35.4.5.	<i>debian/copyright</i>	234
35.4.6.	<i>debian/control</i>	235
35.4.6.1.	Grundlegender Aufbau	235
35.4.6.2.	Anpassungen für Java -Pakete	237
35.4.6.3.	Web-Extension-Plugin	238
35.4.6.4.	Python-Plugin	238
35.4.7.	<i>debian/watch</i>	239
35.4.8.	<i>debian/rules</i> - Grundlegender Aufbau	244
35.4.8.1.	Erstellen der Datei	244
35.4.8.2.	Export von Variablen	244
35.4.8.3.	Aufruf der Debhelper	245

35.4.8.4. <i>debian/rules</i> - overrides	246
35.4.8.5. Schluss der Funktion	246
35.4.9. <i>salsa-ci.yml</i>	247
35.4.10. <i>debian/javabuild</i>	247
35.4.11. <Paketname>.install	248
35.4.12. <Paketname>.dirs	248
35.4.13. <Paketname>.docs	248
35.4.14. <Paketname>.links	249
35.4.15. <Paketname>.desktop	249
35.4.16. <Paketname>.manpages	250
35.4.17. <Paketname>.examples	250
35.4.18. README.Debian	251
35.4.19. README.source	251
35.4.20. <i>source/lintian-overrides</i>	251
35.5. Überprüfung der Dateien in <i>debian/</i> mit CmeFix	251
36. Änderungen am Upstream-Code vornehmen	253
36.1. Sollen Änderungen am Upstream-Code vorgenommen werden?	253
36.2. Arbeiten mit <i>gbp pq</i>	257
36.2.1. Fallunterscheidung	258
36.2.2. Die „einfachen“ Fälle	258
36.2.3. Die „schwierigeren“ Fälle	259
36.2.3.1. Prüfen der Situation	260
36.2.3.2. <i>gbp pq import</i>	262
36.2.3.3. Aktualisieren des Patch-Queue-Zweiges	267
36.2.4. Bearbeiten des Quellcodes	269
36.2.5. Export der Patches	272
36.3. Bauen schlägt fehl	274
36.4. Auswahl der Patches	274
36.5. Nutzung von Quilt	276
36.5.1. Bisher kein Patch vorhanden	277
36.5.2. Was soll getan werden?	279
36.5.3. Neuen Patch erstellen	280
36.5.4. Datei zum Patchen auswählen	281
36.5.5. Patch löschen	285
36.5.6. <i>debian/patches/series</i> editieren	287
36.5.7. Ausgangszustand wiederherstellen	287
36.5.8. Patch bearbeiten	289
37. Bauen	291
37.1. <i>debian/changelog</i>	291
37.1.1. Versionsbezeichnung einfügen	293
37.2. Verschieben der <i>gbp</i> -Konfigurationsdatei	301
37.3. Parameter für <i>gbp buildpackage</i> festlegen	302
37.3.1. Git-Zweig und Distribution ermitteln	302
37.3.2. Git-Zweig und Distribution prüfen	305
37.3.3. Git-Zweig anpassen	306
37.3.4. Distribution ermitteln	307
37.3.5. Auswahl des Build-Systems	307
37.3.6. Überprüfung der Parameter	308
37.3.7. Letzte Ausstiegsmöglichkeit	309
37.4. Was macht <i>sbuid</i> ?	310

37.5.	In der <i>Pbuilder-Chroot</i> bauen	313
37.5.1.	<i>base.cow</i> erstellen	313
37.5.2.	git-pbuilder update	315
37.5.3.	Aufnahme des *.orig-Archives in *.changes	316
37.5.4.	Bauen mit <i>gbp buildpackage</i>	320
38.	Wenn das Bauen fehlschlägt	323
39.	Bauen jenseits von <i>Unstable (sid)</i>	325
39.1.	Bauen für bereits offiziell freigegebene Distributionen	325
39.2.	Proposed-Updates – Besonderheiten	326
40.	Überprüfungen	327
40.1.	Auswahl der Changes-Datei	329
40.2.	Yamllint	330
40.3.	Prüfung mit Lintian	330
40.4.	Uscan	332
40.5.	Überprüfen der Datei <i>debian/copyright</i>	335
40.6.	Überprüfen mit <i>debdiff</i> und <i>diffoscope</i>	335
40.6.1.	<i>debdiff</i>	335
IV.	Veröffentlichen	341
41.	Vorbereitungen zum Hochladen	343
41.1.	Existiert <i>debian/changelog</i> ?	343
41.2.	<i>debian/changelog</i> fertigstellen	345
41.3.	Nochmaliges Bauen?	348
41.4.	Soll ein <i>Debdiff</i> erstellt werden?	349
42.	Hochladen auf Git-Repositoryen	351
42.1.	Hochladen nach salsa.debian.org	351
42.2.	Hochladen auf eigenen Git-Server	355
43.	Paket(e) hochladen	357
43.1.	Auswahl des Zielrepositoriums	358
43.2.	Signatur erzeugen	361
43.3.	Hochladen mit dput	362
43.4.	Nach FTP-Master hochladen	362
43.4.1.	Zurückweisung eines Paketes	367
43.5.	Nach mentors.debian.net hochladen	367
43.6.	Als <i>Non-Maintainer-Upload</i> hochladen	368
43.7.	Hochladen nach <i>people.debian.org</i>	369
43.8.	Lokales Repositoryum	371
V.	Weitere Bestandteile des Skriptes	373
44.	Weitere Aufgaben	375
44.1.	Neuen Branch erstellen	375
44.2.	Eingabe des Namens oder der IP eines eigenen Git-Servers	376
44.3.	AddGitServer	377

45. Kopf des Skriptes	379
45.1. Shebang	379
45.2. Copyright-Vermerk	379
45.3. Abhängigkeiten für das Programmskript	379
45.4. <i>set -e</i> und Funktionsheader	380
45.5. Funktion zur Fehlersuche	380
VI. Plugins und Skripte	381
46. Java-Plugin	383
46.1. Anpassungen für Java-Pakete	383
47. Maven-Plugin	385
47.1. Kopf des Maven-Plugins	385
47.2. Mitteilung	385
47.3. Bauen mit <i>Maven</i>	386
47.4. Maven-Dateien bearbeiten	394
47.4.1. <i>maven.rules</i>	395
47.4.2. <i>maven.ignoreRules</i>	395
47.4.3. <i>maven.properties</i>	396
47.4.4. <i>Paketname.poms</i>	397
47.4.5. <i>README.source</i>	398
47.5. <i>debian/rules</i> - Ergänzungen für Java-Pakete mit <i>Maven</i>	398
48. Web-Extension-Plugin	399
48.1. Kopf des Webext-Plugins	399
48.2. Erstellen der <i>webext*.*</i> -Dateien in <i>debian/</i>	400
48.2.1. Ermittlung des Namens der <i>*.xpi</i> -Datei	400
48.2.2. <i>debian/rules</i> - Ergänzungen für Mozilla-AddOns	401
48.2.3. <i>debian/control</i> - Ergänzungen für Mozilla-AddOns	402
48.2.4. <i>debian/webext-*.install</i>	403
48.2.5. <i>debian/webext-*.docs</i>	403
48.2.6. <i>debian/webext-links.tb</i>	403
48.2.7. Meldung: Webext-Plugin geladen	403
49. Python-Plugin	405
49.0.1. Meldung: Python-Plugin geladen	405
49.1. Anpassungen für Python-Pakete	406
49.2. <i>debian/control</i> - Ergänzung für Python-Pakete	407
50. Skripte	409
50.1. Anlage eines Projektes im Java-Team	409
50.2. Skript zum Extrahieren der Dokumentation im PDF- und Epub-Format	412
50.2.1. Abhängigkeiten	412
50.2.2. Ablauf	412
50.3. Skript zum Extrahieren der Skripte	415
50.4. <i>gitlab-ci.yml</i> für die Salsa-CI	417
VII. Anhang	A–i
Abbildungsverzeichnis	A–iii

4. Mai 2025

Literaturverzeichnis

A–viii

Stichwortverzeichnis

A–xii

Teil I.

Überblick

1. Lizenz

Der Text des Buches „Debian-Pakete bauen mit *Git-Buildpackage*“ von Mechtilde und Michael Stehmann steht unter der **Creative Commons Namensnennung - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (CC BY-SA 4.0)**[1].

Das beschriebene Programmskript steht unter der **GNU General Public License Version 3** oder nach Ihrer Wahl einer späteren Version[2].

Copyright: © 2012-2024 Mechtilde Stehmann (E-Mail: mechtilde@debian.org),
Michael Stehmann (E-Mail: michael@canzeley.de)

2. Zum Titel des Buches

Was ein **Debian-Paket** ist, wird in Kapitel 8 (Seite 23) beschrieben. Vorab sei verraten: Ein **Debian-Paket** ist nicht nur ein Archiv im *deb*-Format, das Binär-Dateien enthält.

git-buildpackage ist ein **Debian-Paket**, welches nützliche Werkzeuge enthält, **Debian-Pakete** aus einem **Git**-Repository zu bauen. Einige dieser Werkzeuge werden vom Programmskript, das in diesem Buch beschreiben wird (Kapitel 31, Seite 105), verwandt (s. Kapitel 19, Seite 57).

Die Namen der Werkzeuge in diesem Paket fangen mit *gbp* an. Ein wichtiges Werkzeug ist *gbp buildpackage*.

3. Wer sollte dieses Buch lesen?

Die Ausführungen in diesem Buch sind besonders für die Nutzer des Skriptes interessant. Aber auch Menschen, die sich allgemein für das Paketieren für eine Distribution interessieren, werden in diesem Buch Informationen finden können.

Dieses Buch will kein „Lehrbuch“ des **Debian**-Paketbaus sein. Es ist eher ein Erfahrungsbericht, wobei die Erfahrungen „in Code gegossen“ worden sind.

Das Buch beschreibt, wie **Debian**-Pakete auf der Basis eines **Git**-Repositoriums mit den Programmen aus dem Paket *git-buildpackage* [3] und anderen nützlichen Befehlen erstellt werden. Am Ende sollte der Leser in der Lage sein, mit der Hilfe des dargestellten Programmskripts und der Beschreibungen der einzelnen Schritte „veröffentlichungsreife“ **Debian**-Pakete zu bauen.

Das Programmskript selbst baut **keine Debian**-Pakete, sondern unterstützt den Nutzer beim Bauen derselben. Es ist nur ein Assistenzprogramm.

Dieses Buch kann auch dazu dienen, Probleme zu verstehen, die beim Paketieren von **Debian**-Paketen auftreten können.

Paketieren ist im Grunde nicht schwierig. Es gibt aber immer wieder neue Herausforderungen. Paketieren macht daher Spaß.

4. Wie entsteht dieses Buch?

4.1. Motivation

Was treibt uns, ein solches Buch zu schreiben?

Dazu muss man folgendes wissen:

Beim Paketieren werden viele Befehle in einer sinnvollen Reihenfolge in der Shell ausgeführt. Außerdem müssen viele kleine Dateien gepflegt und eingebunden werden. Kleinste Fehler und Ungenauigkeiten führen meist dazu, dass das Paket nicht korrekt gebaut werden kann. Auch ist es aufwändig und fehlerträchtig, immer wieder die korrekten Optionen einzusetzen.

Um diese Fehlerquellen möglichst klein zu halten, wurden diese Schritte in einem Shell-Skript zusammengefasst. Im Laufe der Zeit und mit jedem weiteren Paket wächst dieses Skript und wird auch immer weiter verfeinert. Daraus ist bisher schon ein umfangreiches Programmskript geworden.

Als Mechtilde anfang, sich mit dem Bauen von **Debian**-Paketen zu beschäftigen, stand die Frage im Raum, wie sich diese vielen Schritte dokumentieren und automatisieren lassen. Zur Automatisierung ist dann dieses Shell-Skript entstanden. Der Bedarf an Dokumentation konnte von Anfang an nicht mit Kommentaren – weder in den einzelnen Dateien, noch in diesem Programmskript – gedeckt werden.

Deshalb haben wir auch schon früh angefangen, unsere Schritte beim Paketieren ausführlich festzuhalten. Ein besonderes Augenmerk haben wir auf Beschreibungen gelegt, welche getroffene Entscheidungen nachvollziehbar und überprüfbar machen. Dies erleichtert notwendige Veränderungen.

Daher haben wir auch soweit wie möglich bei der Angabe von Optionen die Langform gewählt. Dies erleichtert die Lesbarkeit.

Die Dokumentation soll also sowohl das eigentliche Paketieren beschreiben, als auch das Skript erläutern.

Die **Debian**-Distribution ist das Werk vieler Menschen. Sie besteht aus mehreren zehntausenden Paketen. Das Bauen der Pakete ist eine wesentliche Aufgabe der Paketbetreuer. Viele Paketbetreuer nutzen hierzu eigene Skripte. Die Veröffentlichung eines solchen Skriptes ist daher ein Wagnis. Wenn unser Skript einigen Paketbetreuern das Leben erleichtert und Neulinge an das Paketbauen heranzuführt, hat sich dieses Wagnis gelohnt.

Das beschriebene Programmskript bezieht sich nicht auf ein bestimmtes **Debian**-Paket. Vielmehr sollen damit generell einfache **Debian**-Pakete gebaut werden können.

Es werden die Schritte beschrieben, die wir für das Paketieren der von Mechtilde betreuten Pakete benötigen. Das Programmskript erhebt nicht den Anspruch, man könne damit aus jedem Quellcode ein **Debian**-Paket bauen.

An vielen Stellen kann und muss man auch manuell eingreifen. Hierzu soll die Beschreibung der Vorgänge beim Paketieren eine Hilfe sein.

Dass das Programmskript die Möglichkeit manueller Eingriffe voraussetzt, macht es erforderlich, dass das Programmskript immer wieder prüft, ob die Voraussetzungen, von denen die Verfasser ausgegangen sind, auch tatsächlich vorliegen. Diese Notwendigkeit erhöht leider den Umfang und die Komplexität des Skriptes und damit auch den Umfang des Buches.

4.2. Baustellen

Das Buch und das Skript sind immer noch „Baustellen“, weil immer wieder neue Erfahrungen einfließen.

Das Buch ist in deutscher, die Oberfläche des Programmskripts in englischer Sprache verfasst. Lokalisierungen sind willkommen. Als „Proof-of-Concept“ wurde bereits ein Teil des Buches in die englische Sprache übersetzt.¹

Die Veröffentlichung des Quelltextes erfolgt im vom Debian-Projekt bereitgestellten Git-Repositoryum². Dort ist die CI/CD (Kapitel 50.4, Seite 417) aktiviert. Daher steht das Buch als *.pdf³ und *.epub⁴ zur Verfügung. Dort können auch die Programmskripte heruntergeladen werden⁵

4.3. Werkzeuge

„Schöner Leben mit Dokumentation“ ist ein geflügeltes Wort in unserer *Peergroup*.

Mit welchen Werkzeugen lässt sich eine solche Dokumentation erstellen? Stehen diese Werkzeuge auch als Debian-Pakete zur Verfügung?

Einen wichtigen Hinweis dazu hat Mechtild auf einer Veranstaltung zum **Software Freedom Day 2012**⁶ in Köln erhalten. Dort hat sie die Möglichkeiten von **noweb** [4]⁷ kennengelernt. Dies bedeutete auch, sich mit L^AT_EX näher zu beschäftigen.

In dieser Kombination ist es möglich, Code und dessen Beschreibung in *einem* Dokument zu pflegen. Donald Knuth bezeichnet dies als „Literate Programming“⁸

Weiter haben wir uns damit beschäftigt, dass sich mit L^AT_EX neben PDF-Dokumenten auch Dokumente im EPUB-Format erstellen lassen. Diese können auch mit einem E-Book-Reader gelesen werden. (Kapitel 50.2, Seite 412)

Dieses Buch in eine andere Sprache zu übersetzen, stellt eine besondere Herausforderung da. Unsere Tests haben ergeben, dass **OmegaT**⁹ insoweit ein nützliches und komfortables Werkzeug darstellt. Der entsprechende Prozess wird in einem separaten Büchlein dokumentiert¹⁰.

Das Literaturverzeichnis wird mit *jabref* erstellt und gepflegt. Die so erstellte Datei kann in das L^AT_EX-Dokument eingebunden werden.

Als Editor wird *geany* mit dem Plugin *geany-plugin-latex* verwendet.

Git ist genial. Das Bauen erfolgt daher mit den Werkzeugen aus dem Paket *git-buildpackage*¹¹ (s. Kapitel 19.1, Seite 57).

Die Menschen bei Debian haben viele nützliche Programme geschaffen, die das Bauen von Debian-Paketen erleichtern und vereinheitlichen. Das dargestellte Programmskript wurde daher „auf den Schultern von Riesen“ geschaffen.

Es dient dazu, die eingesetzten Hilfsprogramme in zweckmäßiger Reihenfolge aufzurufen und mit sinnvollen Optionen zu versehen. Es soll seinen Nutzern den Weg weisen

¹https://ddp-team.pages.debian.net/dpb/en_US/BuildWithGBP.pdf

²<https://salsa.debian.org/ddp-team/dpb>

³<https://ddp-team.pages.debian.net/dpb/BuildWithGBP.pdf>

⁴<https://ddp-team.pages.debian.net/dpb/BuildWithGBP.epub>

⁵<https://ddp-team.pages.debian.net/dpb/build-gbp.sh>

<https://ddp-team.pages.debian.net/dpb/build-gbp-maven-plugin.sh>

<https://ddp-team.pages.debian.net/dpb/build-gbp-webext-plugin.sh>

<https://ddp-team.pages.debian.net/dpb/build-gbp-python-plugin.sh>

<https://ddp-team.pages.debian.net/dpb/build-gbp-java-plugin.sh>

⁶https://sfd.koelnerlinuxtreffen.de/SFD2012/2012Robert_Stanowsky.html

⁷s.a. <https://de.wikipedia.org/wiki/Noweb>

⁸<http://www.literateprogramming.com/>

⁹<https://packages.debian.org/sid/omegat>

¹⁰<https://oldmike.pages.debian.net/omegatbooklet/omegat.pdf>

¹¹<https://packages.debian.org/sid/git-buildpackage>

4. Mai 2025

und ihnen die Arbeit erleichtern. Um seine Anpassung an die Bedürfnisse seiner Nutzer zu erleichtern, ist es ein Shellskript.

5. Konventionen

Einige Hinweise zum besseren Verständnis des Buches:

5.1. System

Das Buch und besonders das Programmskript wurden auf einer *64-Bit-PC*- Architektur erstellt. Diese wird bei **Debian** als *amd64* bezeichnet. Eine weitere Bezeichnungen für dieses System ist *x86-64*.

5.2. Terminologie

Ein **neues Paket** ist ein Paket, welches das Programmskript noch nicht kennt. D.h. zu diesem Paket existiert noch keine Konfigurationsdatei.

Eine **neue Version** ist eine neue Upstream-Version. Dem Bauen einer neuen Version folgt der Bau einer neuen Revision.

Eine **neue Revision** bezeichnet ein neu hochzuladenes **Debian**-Paket.

5.3. Typographie

Alle Programmnamen sind in *kursiv* gesetzt. Alle Eigennamen sind in **nichtproportionaler** Schrift gesetzt. Hochgestellte Zahlen weisen auf die Fußnoten auf der gleichen Seite hin. Zitate auf ein Gesamtdokument weisen in eckigen Klammern [] direkt auf das Literaturverzeichnis.

Alle Optionen der Shell-Kommandos werden in der Langform angegeben, soweit dies möglich ist. Dies erhöht die Lesbarkeit.

Für die verwendeten Abkürzungen wird auf die Einträge im Glossar ¹ verwiesen.

5.4. Darstellung des Quellcodes

Die Darstellung des Quellcodes erfolgt in Teilstücken (sogenannten Code Chunks). Die Reihenfolge dieser Code-Teile im Buch entspricht oft nicht der Reihenfolge in den Skripten. Dass die Reihenfolge in Buch und Skript nicht übereinstimmen muss, ist ein Vorzug von **noweb**.

¹<https://wiki.debian.org/Glossary>

6. Kurzanleitung

Dieser Abschnitt soll dem Nutzer helfen, die benötigten Schritte nachzuvollziehen und deren Beschreibung schneller im Buch zu finden.

Neben Neugier, Lernbereitschaft und dem unbändigen Willen, Debian-Pakete zu bauen, benötigt der Maintainer noch zwei Dinge:

- einen Account auf <https://salsa.debian.org> und
- einen GPG-Schlüssel zum Signieren der Pakete.

6.1. Vorbereitung der Build-Umgebung

Bis aus dem Buch und den Skripten ein Debian-Paket entsteht, sind die folgenden Schritte notwendig,

Das Programmskript selbst baut **keine** Debian-Pakete, sondern unterstützt den Nutzer beim Bauen derselben. Es ist nur ein Assistenzprogramm.

1. Herunterladen von salsa.debian.org Der Quelltext dieses Buches und die folgenden beiden Skripte finden sich unter <https://salsa.debian.org/ddp-team/dpb>.
2. Installieren der Abhängigkeiten, um das Buch und die Build-Skripte lokal zu generieren. (Kapitel 50.2.1, Seite 412)
3. Generieren des PDF mit `./create-book.sh` (Kapitel 50.2, Seite 412)
4. Mit `./create-buildscript.sh` wird das Programmskript generiert (Kapitel 50.3, Seite 415).
5. **Alternativ:** Herunterladen der Dateien von
 - <https://ddp-team.pages.debian.net/dpb/BuildWithGBP.pdf>
 - <https://ddp-team.pages.debian.net/dpb/create-book.sh>
 - <https://ddp-team.pages.debian.net/dpb/create-buildscript.sh>
6. Erstellen von Symlinks auf die generierten Skripte unter `/usr/local/bin` Damit liegen die jeweils aktuellen Skripte auch in einem Programmpfad (`PATH`) und können überall ohne Pfadangabe aufgerufen werden.
7. Installation aller Abhängigkeiten, die das Programmskript benötigt. (Kapitel 45.3, Seite 379)
8. **Noch einfacher:** Statt vorstehender Schritte können die Debian-Pakete `debian-packages-book-de` und `debian-package-scripts` aus dem Debian-Repository installiert werden.
9. Anlegen der benötigten Verzeichnisse und Dateien (Kapitel 19.2, Seite 58)
 - Verzeichnis für die Konfigurationsdateien `~/.debian_project/` (Kapitel 19.2.2, Seite 59)
 - Anlegen Projektverzeichnisse und Unterverzeichnisse (Kapitel 19.2.1, Seite 58)
 - Eintragungen in die Datei `~/.bashrc` (Kapitel 19.2.3, Seite 60)
 - Anlegen der Datei `gpb.conf` (Kapitel 20.3, Seite 71)
 - Bei Nutzung des `pbuilder` folgt die Konfiguration (Kapitel 19.3.2, Seite 61) und Einrichtung der Hooks (Kapitel 19.3.3, Seite 64)

6.2. Nutzung des Programmskriptes

1. Bereitstellen des GPG-Keys, um Git-Tags zu signieren
2. Ausführen von *build-gbp.sh* (Kapitel 31.1, Seite 106)
3. Mit dem Projektnamen die Konfigurationsdatei erstellen bzw. laden (Kapitel 31.3, Seite 107).
4. Beschaffen des Upstream-Quellcodes (Kapitel 34, Seite 181)
5. Anlegen bzw. Aktualisieren der Dateien im Verzeichnis *debian/* (Kapitel 35, Seite 225)
6. Änderungen am Quellcode vornehmen (Kapitel 36, Seite 253)
7. Bauen des **Debian**-Paketes (Kapitel 37, Seite 291)
8. Überprüfen der Qualität des gebauten **Debian**-Paketes (Kapitel 40, Seite 327)
9. Veröffentlichen des **Debian**-Paketes (Kapitel 41, Seite 343)

Das Programmskript ist modular aufgebaut, sodass man an vielen Stellen „aussteigen“ und später wieder „einsteigen“ kann. Es ermöglicht auch manuelle Eingriffe.

Vor dem Arbeiten mit dem Programmskript sollte auch in folgenden Teil hineingeschaut werden.

Teil II.
Grundlagen

4. Mai 2025

In diesem Teil des Buches gibt es zunächst ein wenig Theorie. Sodann wird die Einrichtung des Systems einschließlich des Git-Repositorys, welches für *git-buildpackage* essentiell ist, beschrieben.

7. Lektüre

Manche fragen, was sie gelesen haben sollten, bevor sie mit dem Bauen von **Debian**-Paketen beginnen. Andere wollen wissen, wo sie hilfreiche Informationen finden können. Daher gibt es hier Leseempfehlungen für alle, die **Debian**-Pakete bauen (wollen).

Eine kurze Einführung in das Thema „Paketieren für **Debian**“ enthält das **Simple Packaging Tutorial**[5].

Die folgenden drei Dokumente gehören zur „Pflichtlektüre“ eines jeden Paket-Betreibers.

7.1. Debian Free Software Guidelines

Allen, die bei **Debian** mitwirken wollen, wird die Lektüre des Gesellschaftsvertrages empfohlen. Die **Debian Free Software Guidelines (DFSG)**[6] sind wesentlicher Bestandteil des Gesellschaftsvertrages. Sie enthalten die Bedingungen, die eine Lizenz erfüllen muss, um als „frei“ zu gelten. Diese Bedingungen sind bereits bei der Auswahl der zu paketierenden Software bedeutsam (Kapitel 10, Seite 27).

Die **Debian Free Software Guidelines (DFSG)** gelten nicht nur für Software-Lizenzen, sondern gemäß Ziffer 1 der Version 1.1 des Gesellschaftsvertrages („alle Komponenten“) auch für die Lizenzen von Bildern, Tönen, Texten etc.

7.2. Debian Policy-Handbuch

Das **Debian Policy-Handbuch**[7] beschreibt die Richtlinien für die Distribution **Debian GNU/Linux**. Beschrieben werden die Struktur und der Inhalt des **Debian**-Archivs, verschiedene Design-Entscheidungen des Betriebssystems und technische Anforderungen, die jedes Paket erfüllen muss, um in die Distribution aufgenommen zu werden. Dieses Dokument steht nur in englischer Sprache zur Verfügung.

Es steht auch als **Debian**-Paket *debian-policy* bereit.

Der **Filesystem Hierarchy Standard**[8] ist eine wichtige Ergänzung zum Policy-Handbuch.

Daneben gibt es ergänzend noch weitere Regelwerke.¹

7.3. Debian Entwicklerreferenz

In der **Entwicklerreferenz**[9] werden die Verfahren und Ressourcen für **Debian**-Entwickler aufgeführt. Das Dokument beschreibt, wie man ein neuer Entwickler für das **Debian**-Projekt wird, die Upload-Prozedur, wie die Fehlerdatenbank (Bug-Tracking-System), die Mailinglisten, Internet-Server etc. bedient werden.

Dieses Dokument ist als Referenzhandbuch für alle **Debian**-Entwickler gedacht. Es steht als **Debian**-Paket *developers-reference-de* auch in deutscher Sprache zur Verfügung. [9]

¹<https://www.debian.org/devel/index.en.html>

7.4. Referenz für *Git-Buildpackage*

Es gibt verschiedene Verfahren und Werkzeuge für das Bauen von **Debian**-Paketen. In diesem Buch wird *git-buildpackage* verwendet.

Zusätzlich empfehlen wir daher noch die Referenz für das von uns gewählte Build-System *git-buildpackage*[3].

7.5. Handbuch für **Debian**-Betreuer

Dieses Handbuch für **Debian**-Betreuer[10] beschreibt den Bau eines **Debian**-Paketes mittels des *debmake*-Befehls. Es ist für normale Benutzer und angehende Paketbetreuer gedacht.

Der Fokus liegt auf dem modernen Paketierungsstil. Es enthält viele einfache Beispiele.

Dieses „Handbuch für **Debian**-Betreuer“ soll als Erbe des „**Debian**-Leitfaden für Neue Paketbetreuer“ betrachtet werden.

Es steht auch in deutscher Sprache und als **Debian**-Paket *debmake-doc* zur Verfügung. [10]

7.6. **Debian** Leitfaden für Neue Paketbetreuer

Dieses reife Werk[11] versucht, das Erstellen von **Debian**-Paketen für normale Anwender und zukünftige Entwickler verständlich mit funktionierenden Beispielen zu beschreiben.

Anders als frühere Dokumente baut dieses auf *debhelper* und weiteren Werkzeugen, die einem Entwickler zur Verfügung stehen, auf.[11]

Auch dieses Dokument steht in deutscher Sprache und als **Debian**-Paket zur Verfügung.

7.7. Weitere Informationen

Weitere nützliche Literatur findet man unter <https://www.debian.org/doc>.

Zum Thema des Bauens von **Debian**-Paketen kann man im Internet an weiteren Stellen Dokumente finden. Es gilt jedoch zu beachten: „Das Internet vergisst nichts, und irgendwo ist immer noch eine veraltete Dokumentation verlinkt, deren Hinfälligkeit mangels Verfallsdatums auch nicht zu erkennen ist.“²[12]

²Über dieses Buch, Kapitel 2.9 im Buch **Debian**-Paketmanagement

8. Was ist ein Debian-Paket?

Um den Weg zu verstehen, sollte man das Ziel kennen. Das Ziel des Paket-Bauens ist ein **Debian-Paket**[5].

Was aber ist ein **Debian-Paket**?

Eine formale Antwort lautet: Ein **Debian-Paket** ist ein Software-Paket, welches vom **Debian-Projekt** veröffentlicht wurde.

Damit ein Paket vom **Debian-Projekt** veröffentlicht wird, muss es Anforderungen genügen, die vom **Debian-Projekt** aufgestellt, schriftlich niedergelegt und zwecks Transparenz veröffentlicht worden sind.

Ein **Debian-Paket** ist ein Software-Paket, welches vor allem den Anforderungen genügt, die in der **Debian-Policy**[7] beschrieben werden. Es gehört zur vom **Debian-Projekt** gepflegten Transparenz, dass die genaue Version der **Debian-Policy**, die beim Bauen des Paketes beachtet wurde, in der Datei *debian/control* angegeben wird. (Kapitel 35.4.6, Seite 235)

Ein **Debian-Paket** ist eine Sammlung von Dateien, die es ermöglichen, Anwendungen oder Bibliotheken über das **Debian-Paketverwaltungssystem** zu verteilen. Das Ziel der Paketierung ist es, die Automatisierung der Installation, der Aktualisierung, der Konfiguration und des Entfernens von Software für **Debian** auf eine konsistente Weise zu ermöglichen.[5][13]

Ein **Debian-Paket** ist mehr als nur eine Archiv-Datei mit ausführbarem Code mit der Endung *.deb*. Ein **Debian-Paket** besteht aus insgesamt **vier** Dateien; davon sind drei Archive.

1. Eine Archiv-Datei mit der Endung *.orig.tar.gz* bzw. *.orig.tar.xz* enthält den Quellcode, aus dem das Paket gebaut wird.
2. Eine weitere Archiv-Datei mit der Endung *.debian.tar.xz* enthält die debian-spezifischen Dateien, die den Bauvorgang und die Installation steuern oder zusätzliche Informationen enthalten.
3. Eine Datei enthält neben beschreibenden Angaben zum Paket die Prüfsummen der beiden vorgenannten Archiv-Dateien. Diese Datei hat die Endung *.dsc*. Diese Datei ist signiert.
4. Den ausführbaren Code enthält schließlich das Archiv mit der Endung *.deb*. Auch diese Datei ist signiert.

In der Regel werden nur die erstgenannten drei Quellcodedateien vom Maintainer hochgeladen. Die Datei, die den ausführbaren Code enthält, wird aus den Quellcodedateien auf der Projekt-Infrastruktur gebaut.

Das **Debian-Paket-System** ermöglicht die Nachvollziehbarkeit vom Upstream-Quellcode bis zum binären **Debian-Paket**. Angestrebt und auch oft erreicht wird eine bitgenaue Reproduzierbarkeit des Bauprozesses[14].

Wie dies nachgeprüft werden kann, wird in Kapitel 29 (Seite 99) beschrieben.

Diese Transparenz gibt dem Anwender Sicherheit, dass das Binär-Paket auch nachvollziehbar aus dem veröffentlichten Quellcode gebaut wurde.

Wer Software aus seinem Build-System in irgendein Archiv im *deb*-Format speichert, ohne sich um Standards und Regeln zu kümmern, packt kein **Debian-Paket**.

Vom **Debian-Projekt** werden zu jedem Zeitpunkt mehrere Varianten (Releases) angeboten, nämlich in erster Linie *stable*, *testing* und *unstable*. Nach der Veröffentlichung jeder neuen *Stable*-Version wird die bisherige *Stable*-Version noch einige Zeit lang

als *Oldstable* gepflegt. Desweiteren gibt es noch *Oldoldstable* und einen Zweig *Experimental*. Dort werden Änderungen ausprobiert, die gravierende Auswirkungen auf das Gesamtsystem haben könnten.

Der Zweig *experimental* ist allerdings keine komplette Distribution, sondern funktioniert nur als Erweiterung von *unstable* [15].

Der Zweig *unstable* (*sid*) ist der erste Anlaufpunkt für neue Programme und neue Versionen von Programmen, bevor sie in *testing* integriert werden. Die Entwicklung eines *Debian*-Paketes beginnt somit in der Regel im Zweig *unstable*.¹

Jedes *Debian*-Paket gehört zu einem definierten Entwicklungsstadium der *Debian*-Distribution, also zu einer bestimmten, veröffentlichten Version. Diese werden als Releases bezeichnet.²

Jedes Paket muss auf diese Veröffentlichung abgestimmt sein. Es darf keine Abhängigkeiten zu einer anderen Veröffentlichung haben. Bibliotheken dürfen in einer Veröffentlichung grundsätzlich nur in einer Version vorhanden sein, damit Sicherheitsaktualisierungen für den Anwender einfach möglich sind. Ein *Debian*-Paket darf also keine eigene Version einer solchen Bibliothek enthalten.³

¹DPMB, I.Konzepte, Kapitel 2.10.1[12]

²DPMB,Kapitel 2.10 in [12]

³Kapitel 7.1 [16]

9. Auswahl der zu paketierenden Software

Oft wird die Frage an das Projekt gestellt, welches Paket sich denn zum Starten eigne. Zum Erlernen des Paketierens gibt es kein eigens dafür erstelltes „Schulungs-“Paket. Vielmehr erlernt man das Paketieren durch das Packen von **Debian**-Paketen („Learning by doing“).

Es wird von Anfang an darauf Wert gelegt, dass die Motivation mitgebracht wird, sich Schritt für Schritt in die Thematik des Paketierens einzuarbeiten. Eine gute Voraussetzung dafür ist auch, dass es eine Software ist, die man gerne selber als **Debian**-Paket nutzen und anderen zur Verfügung stellen möchte.

Oft gibt es auch Pakete, für die die Maintainer Hilfe benötigen oder die verwaist sind. Dies kann auch auf selbstgenutzte Pakete zutreffen. Um herauszufinden, ob und welche der installierten Pakete betroffen sind, gibt es ein Werkzeug, das man sich zusätzlich als Paket *how-can-i-help* [17] installieren kann.

Diese Installation führt dazu, dass *apt* am Schluss *how-can-i-help --apt* aufruft. Damit werden dann hilfebedürftige Pakete aufgelistet, die gerade in der lokalen Installation aktualisiert wurden.

Mit *how-can-i-help --old* kann man sich anzeigen lassen, welche der installierten Pakete Hilfe benötigen.

Mit *how-can-i-help --all* werden alle Pakete angezeigt, für die Hilfe benötigt wird¹.

Unter <https://www.debian.org/devel/wnpp/> findet man auch diese Informationen. Das Akronym *wnpp* kürzt *Work-Needing and Prospective Packages* ab und steht sinngemäß für Pakete, an denen Arbeit erforderlich ist und die im Entstehen sind (angehende Pakete).

Motivierend und hilfreich ist es auch, wenn man im Upstream-Projekt aktiv ist. In diesen Fällen ist eine gewisse Nachhaltigkeit der Betreuung des Paketes gewährleistet.

Für viele Kategorien von Paketen haben sich Betreuer-Teams gebildet². Wählt man ein Paket, welches in das Portfolio eines solchen Teams passt, hat man bessere Chancen, Unterstützung und Sponsoren zu finden. Unterstützung findet man auch auf den Mailinglisten der jeweiligen Teams.

Bevor man mit dem Paketieren anfängt, sollte man natürlich prüfen, ob die Software bereits paketiert ist. Mit *apt search <name>* kann festgestellt werden, ob es bereits ein entsprechendes **Debian**-Paket gibt. Auf salsa.debian.org kann nachgeforscht werden, ob schon Paketerversuche im Gange sind.

Für alle Fragen zum Paketieren für **Debian** gibt es auch eine spezielle Mailingliste³. Schon das Mitlesen dieser Liste ist sinnvoll.

¹Weitere Einzelheiten zu *how-can-i-help* werden in *Debian-Paketmanagement*[12] in Kapitel 37.3.6 beschrieben.

²Eine Liste der Teams findet sich im Wiki[18]

³<https://lists.debian.org/debian-mentors/>

10. Prüfung der Quellen

Nach der Auswahl der für **Debian** zu paketierenden Software muss der Quellcode näher untersucht werden. Ziel dieser Untersuchung ist die Feststellung, dass die ausgewählte Software vom Paketbetreuer für **Debian main** paketierte werden kann. Dazu ist zu prüfen, ob **alle** Teile des Quellcodes konform zu den **Debian Free Software Guidelines (DFSG)**[6] und mit der **Debian-Policy**[7] vereinbar sind.

Dies erfordert eine intensive Prüfung **aller** Quellcode-Dateien.

10.1. Lizenzprüfung

In **Debian** darf im Zweig **main** ausschließlich Freie Software im Sinne der **Debian Free Software Guidelines** [6] veröffentlicht werden. Dabei ist zu beachten, dass die **Debian Free Software Guidelines (DFSG)** nicht nur für Software-Lizenzen gelten, sondern gemäß Ziffer 1 der Version 1.1 des Gesellschaftsvertrages („alle Komponenten“) auch für die Lizenzen von Bildern, Tönen, Texten etc.

Die Prüfung dieser Anforderung hat einen großen Anteil an der Arbeit eines Paket-Betreibers, besonders um ein neues Paket in **Debian** zu veröffentlichen.

Bei der Erstellung und Pflege der *debian/copyright*-Datei wird dann diese Arbeit verwendet. Damit ist diese Datei auch eine der wichtigsten Dateien eines **Debian**-Paketes. Sie beschreibt die urheberrechtliche Situation.

Diese Datei muss das Copyright und die Lizenzen aller Dateien eines Quellpakets genau unter Verwendung einer spezifischen Syntax beschreiben (DEP-5).[19]

Die *debian/copyright*-Datei wird von den FTP-Mastern[20] überprüft, wenn ein neues Paket im **Debian**-Projekt angenommen werden soll.

Um zu vermeiden, dass hierzu Einträge übersehen werden, gilt mindestens ein Vier-Augen-Prinzip (Peer-Review)[21].

Der Inhalt der Copyright-Datei muss also die Lizenzen aller Dateien genau wiedergeben. Die Lizenz wird oft in den Kommentaren einer Quelldatei angegeben.

Der gesamte Quellcode ist unter diesem Aspekt zu prüfen. Es gibt einige Hilfsmittel, die dem Paket-Betreuer helfen, diese Tests durchzuführen. ¹

10.1.1. *debmake*

Der Befehl *debmake -cc* wird auch vom Programmskript zur Erstellung der Datei *debian/copyright* genutzt (Kapitel 35.4.5, Seite 234).

In der Praxis zeigt sich, dass dies manchmal nicht ausreicht. Dies liegt daran, dass die Informationen zu den Lizenzen und den Urhebern nicht einem Standard folgend abgelegt sind. Daher findet das Programm *debmake* nicht alle Einträge bezüglich des benötigten *Copyrights*. Es werden also weitere Tests benötigt.

10.1.2. *licensecheck*

Das Programm *licensecheck* (aus dem gleichnamigen Paket) ist in der Lage, Quelldateien zu scannen und meldet das Copyright und die darin angegebenen Lizenzen. Er fasst

¹<https://wiki.debian.org/CopyrightReviewTools>

4. Mai 2025

diese Informationen jedoch nicht zusammen: Für jede Datei eines Pakets wird eine Copyright-Zeile generiert.

Der Test wurde mit der Befehlszeile

```
licensecheck --check '*' --recursive \  
--deb-machine --lines 0 -- *
```

wie im Wiki² ausgeführt.

Leider ist die Handbuchseite (Manpage) von *licensecheck* eher unergiebig. Mehr Informationen liefert der Befehl:

```
licensecheck --help
```

Die Ausgabe muss manuell ausgewertet werden, da *licensecheck* auch versucht, das Copyright für sogenannte Binärdateien (z. B. Bilder, Fonts) anzuzeigen.

10.1.3. *scan-copyrights*

Mit dem Programm *scan-copyrights* aus dem Paket *libconfig-model-dpkg-perl* kann eine vorhandene Copyright-Datei durch erneutes Scannen der Quelle aktualisiert werden.

Die Befehlszeile dazu lautet:

```
scan-copyrights
```

Das Programm kann auch eine solche Datei von Grund auf neu erstellen. Dies erfolgt im DEP-5-Format. [19]

Dieses Programm ist in Perl geschrieben und nutzt *licensecheck*.

10.1.4. *licensing*

Der Befehl *licensing* aus dem Paket *licenseutils* kann den Quellcode scannen und gefundene Lizenzen mit dem Befehl

```
licensing detect *
```

melden. Es kann auch Lizenz-Vorlagen zu neuem Code hinzufügen.

10.1.5. *cme*

Ein weiteres Tool ist *cme*. *cme* prüft und/oder editiert die Daten in den Konfigurationsdateien. Damit kann man unter anderem prüfen, ob die vom Original-Autor bereitgestellte Copyright-Datei alle notwendigen Informationen enthält.

Der Befehl *cme fix dpkg* prüft die *dpkg*-Dateien, aktualisiert veraltete Parameter und wendet alle Korrekturen an (Kapitel 35.5, Seite 251).

Mit

```
cme update dpkg-copyright
```

werden die in den Headern der Quellcodedateien aufgeführten Lizenzen geprüft und aufgelistet.

Mit

```
cme check dpkg-copyright -file <Pfad/Dateiname>
```

können Daten aus einer beliebigen Datei gelesen werden.

Zu diesem Paket gibt es mit *libcomcnfig-model-tkui-perl* auch eine graphische Oberfläche.

²<https://wiki.debian.org/CopyrightReviewTools#licensecheck>

10.1.6. Manuell

Der Gebrauch dieser Werkzeuge ist manchmal nicht ausreichend. Es kann vorkommen, dass irgendwo im Code weitere Autoren genannt werden. Hiernach kann mit folgenden Befehlen gesucht werden:

1. `grep --recursive --ignore-case "(c)" .`
2. `grep --recursive --ignore-case "copyright" .`
3. `grep --recursive --ignore-case "author" .`

10.1.7. Stolperfallen

Es gibt folgende Stolperfallen:

1. mitgelieferte Build-Abhängigkeiten
2. Microcode
3. mitgelieferte unfreie Dokumente
4. vom Upstream-Autor unvollständig beachtete Lizenzen

Solche Dateien müssen aus dem zu veröffentlichenden **.orig.tar.xz* ausgeschlossen werden, sofern dies sinnvoll möglich ist. Dabei ist auch eine entsprechende Versionsbezeichnung zu wählen. [22]. In diesem Programmskript ist vorgesehen, die Dateien mithilfe der Datei *debian/copyright* auszuschließen (Kapitel 34.3.5, Seite 197)

Wenn im Quellcode-Paket unterschiedlich lizenzierte Dateien enthalten sind, ist neben der Prüfung der Lizenzen der einzelnen Quellcode-Dateien auch das Quellcode-Paket insgesamt zu untersuchen. Es ist zu prüfen, ob die verwendeten Lizenzen miteinander kompatibel sind. Hierbei ist *FINOS Open Source License Compliance Handbook*[23] hilfreich.

10.2. Feststellung der Programmiersprache

Software wird in unterschiedlichen Programmiersprachen geschrieben. Es gibt auch Programme, die in mehreren Programmiersprachen geschrieben sind. Je nach Programmiersprache sind verschiedene Compiler- und Build-Systeme zu verwenden.

Für das Bauen der **Java**-Pakete gibt es mehrere Build-Systeme. Dies wird in einem eigenen Kapitel ausführlich beschrieben (Kapitel 13, Seite 39).

10.3. Prüfung der Abhängigkeiten

Es wird zwischen zwei Arten von Abhängigkeiten unterschieden:

- Software, die für das Bauen des Paketes notwendig ist (build dependencies),
- Software, die für die Installation und/oder den Ablauf des paketierten Programms erforderlich ist (runtime dependencies)

Um ein Paket in **Debian main** veröffentlichen zu können, müssen **alle** Abhängigkeiten (inklusive Build-Abhängigkeiten) bereits in **Debian main** verfügbar sein[7]. Dies bedeutet, dass Abhängigkeiten, die noch nicht verfügbar sind, zunächst zu paketieren sind.

Wie die Abhängigkeiten ermittelt werden können, hängt von der Programmiersprache ab. Hinweise auf unerfüllte Build-Abhängigkeiten findet man auch in den Fehlermeldungen beim Bauen eines Paketes. Angaben dazu, warum das Bauen fehlgeschlagen ist findet man in der Datei *<Paketname>_<Version>.build*.

Für **Java**-Programme gibt es verschiedene Orte, an denen diese Informationen gefunden werden können. Dies wird im entsprechenden Kapitel beschrieben (Kapitel 13, Seite 39)

Die dann gefundenen Build- und Runtime-Abhängigkeiten müssen in der Datei *debian/control* im passenden Abschnitt aufgeführt werden (Kapitel 35.4.6, Seite 235).

10.3.1. Abhängigkeiten ermitteln mit *packages.debian.org*

Zur Ermittlung, ob die benötigte Abhängigkeit bereits paketiert ist, kann die Webseite *packages.debian.org* eine Anlaufstelle sein.

Im Abschnitt *Durchsuchen der Paket-Verzeichnisse* sind die Optionen *Nur Paketnamen* und *Quellpaket-Namen* interessant.

Im Abschnitt *Durchsuchen des Inhalts von Paketen* ist es dann die Option *Pakete mit Dateien, deren Namen das Suchwort enthalten*.

Als Distribution sollte dann *alle* bzw. *Unstable* ausgewählt werden.

10.3.2. Abhängigkeiten ermitteln auf *codesearch.debian.net*

Manchmal kann es hilfreich sein, nach Quellcode zu suchen, in dem ein ähnliches Problem schon mal gelöst wurde.

Ein Beispiel ist, dass in einem Paket, das mit *Maven* gebaut wurde, schon einmal ein bestimmter Eintrag in der Datei *debian/maven.rules* erfolgte.

Eine solche Suche kann – wie folgt – durchgeführt werden:

```
https://codesearch.debian.net/search?q=<SearchString>
```

10.3.3. Suche nach Abhängigkeiten auf der Konsole

Mit *apt-file search*³ (Alias: *apt-file find*) kann festgestellt werden, ob benötigte Abhängigkeiten bereits in Debian paketiert sind. Mit *apt-file list <Paketname>* (Alias: *apt-file show*) kann der Inhalt eines Paketes aufgelistet werden. Hier kann dann nach Programmnamen gesucht werden. Die gleichen Informationen erhält man mit *dpkg -L <Paketname>*.

10.4. Änderungen am Quellcode

Der Quellcode ist sorgfältig darauf zu prüfen, ob Änderungen an ihm für das *Debian*-Paket vorgenommen werden müssen. Eine solche Notwendigkeit kann verschiedene Ursachen haben.

Das *Upstream-Release* kann Teile enthalten, die nicht der *Debian-Policy*[7] oder den *Debian-Free-Software-Guidelines*[6] entsprechen.

Änderungen am Quellcode speziell für das *Debian*-Paket können wie folgt vorgenommen werden:

- Es können ganze Dateien ausgeschlossen werden. Dies geschieht beim Bauen einer neuen Version (Kapitel 34.3.5, Seite 197).
- Es können Änderungen an *Upstream-Dateien* durch *Patches* (Kapitel 10.4.2, Seite 32) vorgenommen werden. Dies erfolgt durch das Programmskript beim Bauen einer neuen Revision (Kapitel 36, Seite 253).

10.4.1. Ganze Dateien ausschließen

Wenn ganze Dateien ausgeschlossen werden müssen, muss ein Quellcode-Paket erstellt werden, das diese Dateien nicht mehr enthält.

³<https://manpages.debian.org/unstable/apt-file/apt-file.1.en.html>

Bevor *mk-origtargz* aufgerufen wird, ermöglicht das Programmskript einzelne Quellcode-Dateien von der Aufnahme in das *orig*-Archiv auszuschließen.

Die Dateien, die nicht der Debian-Policy[7] oder den Debian-Free-Software-Guidelines (DFSG)[6] entsprechen, sind zu entfernen.

Dabei ist immer eine neue Version zu bauen, auch dann, wenn kein neues Upstream-Quellcode-Archiv verwendet wird.

Die Ausschlüsse sollen mit ihrer Begründung in der Datei *debian/README.source* (Kapitel 35.4.19, Seite 251) dokumentiert werden.

10.4.1.1. Auflistung der auszuschließenden Dateien

Die auszuschließenden Dateien können entweder in einer separaten Datei oder in der Datei *debian/copyright* im *DEP-5-Format* [19] aufgelistet werden.

Beispiel:

```
Format: https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: <UpstreamName>
Source: <URL to upstream source>
Files-Excluded: <all files to be excluded>
Comment: <description, why the files are excluded>
```

In der Aufzählung werden die Dateien, die auszuschließen sind, durch Leerzeichen oder Zeilenumbruch mit Einrückung um ein Zeichen separiert.

10.4.1.2. Fallunterscheidungen

Oft steht schon vor der Einreichung des Paketes zur *New Queue* fest, dass Dateien auszuschließen sind. Da es zu diesem Zeitpunkt noch keine Datei *debian/copyright* gibt, bietet es sich an, die auszuschließenden Dateien in einer separaten Datei aufzulisten.

Auf die entsprechende Frage im Programmskript ist anzugeben, dass Dateien auszuschließen sind. Dann ermittelt das Programmskript, wo die Informationen zum Ausschluss von Dateien hinterlegt werden sollen. Dies ist vor der Einreichung des ersten Paketes eines neuen Projektes eine separate Datei.

Dazu wird eine Datei *<SourceName>-excluded* im Verzeichnis *PrjPath* (Kapitel 32.2.2, Seite 130) erstellt.

Die Informationen in dieser Datei sind bei der Erstellung der Datei *debian/copyright* dorthin zu übertragen (Kapitel 35.4.5, Seite 234).

Nach einer Zurückweisung aus der *New Queue* muss das Paket neu für eine Veröffentlichung bereit gestellt werden. Auch hierbei kann es vorkommen, dass Dateien, deren Lizenz nicht DFSG-konform ist, ausgeschlossen werden müssen. Der Ausschluss der Dateien gilt bereits für das zu veröffentlichende **.orig.tar.(gz | bz2 | xz)*. Meist steht zu diesem Zeitpunkt aber keine weitere (neuere) Upstream-Version zur Verfügung. Die zu erstellende Version muss jedoch größer sein als die bisherige Version aber kleiner als die nächste zu erwartende Upstream-Version.

Im Falle des Entfernens von Dateien, die nicht DFSG-konform sind, ist dies der Anhang *+dfsg*, der sich direkt an die Upstream-Version anschließt. Ist dieser Anhang bereits vorhanden, so wird er nach *+dfsg1* hochgezählt (s. Kapitel 10.4.1.3, Seite 32). So kann mit *mk-origtargz* ein neuer Tarball mit einer größeren Versionsbezeichnung erstellt werden. (Kapitel 11.2, Seite 35)

Die Namen der auszuschließenden Dateien werden der dann schon vorhandenen Datei *debian/copyright* im DEP-5-Format[19] hinzugefügt. Nach dem Commit dieser Anpassungen im Verzeichnis *debian/* kann eine neue Version gebaut werden. Dazu wird das bisherige Upstream-Archiv verwandt.

Auf diese Weise wird ein neuer Orig-Tarball mit *mk-origtargz* ohne die auszuschließenden Dateien aus dem bisherigen **.tar.gz* erstellt und dessen Inhalt mit *gbp import-orig* in das vorhandene Git-Repository eingefügt (Kapitel 34.3.11, Seite 218).

4. Mai 2025

Wird es bei einer neuen Upstream-Version notwendig, dass Dateien auszuschließen sind, oder ändern sich die auszuschließenden Dateien, ist genauso zu verfahren, wie soeben dargelegt. Es muss die Copyright-Datei geändert und ein entsprechender Commit ausgeführt werden. Wiederum ist beim Bauen der neuen Version anzugeben, dass Dateien auszuschließen sind.

10.4.1.3. Benennung der Pakete beim Ausschluss von Dateien

Damit die Versionsbezeichnung dieser neuen Version größer als die der bisherigen, aber kleiner als die zu erwartenden nächsten Upstream-Version ist, wird die bisherige Versionsbezeichnung mit einem Anhang versehen. Dieser macht zugleich deutlich, dass und warum der Upstream-Code verändert wurde.

Übliche Ergänzungen der Versionsbezeichnung sind *+dfsg* und *+ds*⁴.

Wenn Lizenzen einzelner Dateien des Upstream-Quellcodes nicht den **Debian Free Software Guidelines** (DFSG) genügen und daher nicht als Quellen durch Debian veröffentlicht werden sollen, wird als Ergänzung *+dfsg* benutzt. Erfordern andere Gründe den Ausschluss, nimmt man *+ds* („Debian Source“).[22]

Es kann nämlich passieren, dass zunächst nur Dateien ausgeschlossen wurden, die nicht der Debian-Policy entsprechen.

Wird später festgestellt, dass das hochgeladene Original-Archiv noch unfreie Dokumente enthält, führt dies dann zu einer Versionsbezeichnung nach dem Muster *<Versionsnummer>+ds+dfsg*⁵

In manchen Fällen kann ein Pluszeichen (+) Probleme vor allem beim Bauen von Java-Paketen verursachen.

Die dargestellte Benennung der Debian-Pakete hat auch Konsequenzen für den Inhalt der Datei *debian/watch* (Kapitel 11.4 Seite 36 und Kapitel 40.4 Seite 332).

10.4.2. Änderungen in einzelnen Quellcode-Dateien (Patches)

Änderungen an Quellcode-Dateien sind dann notwendig, wenn aufgrund unterschiedlicher Build-Umgebungen Anpassungen vorzunehmen sind. Andere Fälle sind Fehlerbehebungen, vor allem *Security-Patches*.

Die klassische Methode ist die im Folgenden beschriebene mit *quilt*.

Daneben kann dies auch in einem *Patch-Queue*-Zweig mit *gbp pq* erfolgen (Kapitel 10.4.2.2, Seite 33)

10.4.2.1. Patches mit *quilt*

Änderungen an Quellcode-Dateien erfolgen durch sogenannten Patch-Dateien. Diese dokumentieren den ursprünglichen Quellcode und die jeweiligen Änderungen. Eine weitere Datei (*debian/patches/series*) steuert die Reihenfolge der Anwendung der Patches. Herkömmlicherweise geschieht dies mit *quilt*. *Dquilt* ist eine **debian**-spezifische Anpassung für *quilt*.

Hierzu gibt es eine Beschreibung im *Debian-Leitfaden für Neue Paketbetreuer*⁶ und im *Debian-Wiki*[24]. Daneben gibt es eine allgemeine Einführung in die Nutzung von Quilt [25].

In Kapitel 19.6 (Seite 69) wird beschrieben, wie die **debian**-spezifische Anpassung erzeugt wird.

⁴<https://wiki.debian.org/DebianMentorsFaq> in section *What does „dfsg“ or „ds“ in the version string mean*

⁵s. Mentors-FAQ 2.6[22]

⁶<https://www.debian.org/doc/manuals/maint-guide/modify.de.html>

10.4.2.2. Patchen in einem *Patch-Queue-Zweig*

Da ein Arbeitsablauf mit `Git` bzw. `git-buildpackage` [3] zum Bauen der Debian-Pakete genutzt wird, bietet es sich an, `gbp pq` statt `quilt`⁷ einzusetzen.

Die Änderungen erfolgen dann in einem eigenen `Git`-Zweig

Bei der Nutzung von `gbp pq` erfolgen in diesem Zweig dann *alle* Codeänderungen. Diese Änderungen sollten thematisch und kleingliedrig sein. Dann können die Patches einfacher nach Upstream fließen oder von anderen Distributionen übernommen werden. Dies erleichtert auch die Anpassung an eine neue Upstream-Version.

Die Grundidee ist, dass Patches vom `Debian`-Zweig in den `Patch-Queue-Zweig` so importiert werden, dass jeweils eine Patch-Datei in `debian/patches` jeweils einem Commit auf dem `Patch-Queue-Zweig` entspricht.

Die Anlage des `Patch-Queue-Zweiges` erfolgt mit `gbp pq import`. (s. a. Kapitel 36.2.2, Seite 258)

Der erstellte Zweig wird nach dem Zweig benannt, aus dem er importiert wurde. Zur Unterscheidung wird `patch-queue/` vorangestellt. Wird also die `Debian`-Paketierung auf `debian/sid` gemacht und ein `gbp pq import` ausgeführt, wird der neu erstellte Zweig `patch-queue/debian/sid` genannt.

Das Programmskript ermöglicht diesen Import (Kapitel 36.2.2, Seite 258).

Im `Patch-Queue-Zweig` kann mit den bekannten `Git`-Befehlen (`rebase`, `commit` und `--amend`, etc.) an den Commits gearbeitet werden. Wenn dies erledigt ist, wird `gbp pq export` verwendet, um die Commits auf dem `Patch-Queue-Zweig` wieder in Patches in `debian/patches/`-Dateien umzuwandeln (Kapitel 36.2, Seite 257).

Dieser beschriebene Workflow erleichtert z. B. das Cherry-Picking von Patches für stabile Releases, die Weiterleitung von Patches an neue Upstream-Versionen durch die Verwendung von `git rebase` auf den `Patch-Queue-Zweig` (bereits angewandte Patches werden automatisch erkannt) sowie das Neuordnen, Ablegen und Umbenennen von Patches, ohne auf `quilt` zurückgreifen zu müssen. Die erzeugten Patches in `debian/patches/` haben alle notwendigen Informationen, um sie nach Upstream weiterzuleiten, da sie ein Format ähnlich dem `git-format-patch` verwenden.

Der Hauptnachteil dieses Arbeitsablaufes ist der Mangel an Historie im `Patch-Queue-Zweig`, da er häufig fallen gelassen und neu erstellt wird. Aber es gibt natürlich eine vollständige Historie auf dem `Debian`-Zweig im Verzeichnis `debian/patches/`.

Zu beachten ist, dass `gbp pq` derzeit keine vollständige Unterstützung für `DEP3-Header`[26] bietet.

Zunächst wird versucht, mit `git-mailinfo`[27] zu parsen, was nur die Felder `From` und `Subject` unterstützt. Wenn keines dieser beiden vorhanden ist, wird `gbp pq` versuchen, den Patch vom `DEP3`-Format in ein `git-mailinfo(1)`-kompatibles Format zu konvertieren. Dabei wird zuerst aus den Feldern "Autor" und "Betreff" über die erste Zeile des Feldes Beschreibung geladen. Anschließend werden alle zusätzlichen Felder (wie "Origin" und "Forwarded") und der Rest der Beschreibung (falls vorhanden) an den `mail body` angehängt.

⁷section gbp.patches.html [3]

11. Versionierung der Pakete

Sowohl die Software, die für Debian gebaut werden soll, als auch **Debian**-Pakete haben Versionsbezeichnungen. Dabei sollte die Versionsbezeichnung der Upstream-Software in der des Debian-Paketes enthalten sein.

Die Namen aller Debian-Binärpaketdateien sind folgendermaßen aufgebaut:

`<foo>_<Versionsnummer >-<Debian-Revisionsnummer>_ <Debian-Architektur>.deb.` ¹

11.1. Paketname

Vor der Versionsbezeichnung kommt der Paketname.

Dieser muss gänzlich in Kleinbuchstaben geschrieben werden. Die Versionsbezeichnung kann auch Ziffern enthalten. Zusätzlich können +, -, ~ enthalten sein.

Die Versionsbezeichnung muss mit einer Ziffer beginnen.

11.2. Versionierungsschema

Die Versionierung der Pakete folgt einem fest definierten Schema. Damit wird sichergestellt, dass alle Werkzeuge, die diese Versionierung verwenden, auf die gleiche Systematik zugreifen können. Dabei muss sichergestellt werden, dass das System die neue Versionsbezeichnung auch als größer als die vorherige Versionsbezeichnung erkennt. (s.a Kapitel 34.3.4, Seite 193)

Wie muss also die Versionsbezeichnung des zukünftigen Paketes zusammensetzt sein? Nach der Versionsbezeichnung des Upstream-Paketes können debian-spezifische Zusätze erfolgen. Diese können anzeigen, ob und warum Teile der Upstream-Software entfernt wurden oder ob das Paket auf ein älteres Paket zurückportiert wurde. Außerdem ist in der Versionsbezeichnung des Paketes eine **Debian**-Revisionsnummer enthalten. Diese Revisionsnummer wird insbesondere dann erhöht, wenn das Paket wegen Korrekturen in Dateien im Verzeichnis *debian/* neu gebaut wurde.

Mögliche Varianten können mit dem folgenden Befehl auf ihre Tauglichkeit überprüft werden:

```
dpkg --compare-versions Version1 Operant Version2 && \  
echo "OK"
```

Als Operanden sind diese zu nutzen: lt le eq ne ge gt

lt kleiner als (<)

le kleiner oder gleich (<=)

eq gleich (=)

ne ungleich (!=)

ge größer oder gleich (>=)

gt größer als (>)

Die Versionsvergleichsregeln können wie folgt zusammengefasst werden:²

- Zeichenketten werden von Anfang bis Ende verglichen.

¹DebianFAQ2019, Kapitel 7.3[16]

²New Maintainer Guide, Kapitel 2.6 [11, Kapitel 2.6] s.a. Debmake-doc, Kapitel 5.2

- Buchstaben sind größer als Ziffern.
- Ziffern werden als Ganzzahlen verglichen.
- Buchstaben werden in ASCII-Sortierreihenfolge verglichen.
- Es gibt besondere Regeln für Punkt (.), Plus- (+) und Tilde- (~) Zeichen, wie folgt: *Beispiel*: $0.0 < 0.5 < 0.10 < 0.99 < 1 < 1.0\sim rc1 < 1.0 < 1.0+b1 < 1.0+nmul < 1.1 < 2.0$

In manchen Fällen kann ein Pluszeichen (+) Probleme vor allem beim Bauen von Java-Paketen verursachen.

Die “richtige” Versionierung spielt eine Rolle, wenn diese von Programmen gelesen und genutzt werden soll. Dies sind vor allem die Programme *dpkg*, *apt* und *uscan*.

11.3. *apt* und *dpkg*

Die korrekte Versionierung des Debian-Paketes ist dann nicht ganz einfach, wenn nicht für Debian-Sid gebaut wird (Kapitel 22.6, Seite 83).

dpkg bzw. *apt* müssen zutreffend ermitteln können, ob ein neueres Paket im jeweiligen Release-Zweig zur Verfügung steht.

11.4. *uscan* und die Datei *debian/watch*

uscan ist ein nützliches Werkzeug im Debian-Projekt. Mittels *uscan* kann geprüft werden, ob es eine Upstream-Version gibt, die aktueller (also höher versioniert) ist, als das aktuelle Debian-Paket.

Das Programm *uscan* wird mit dem Paket *devscripts* zur Verfügung gestellt.

Anhand der in *debian/watch* enthaltenen Daten prüft *uscan*, ob neuere Versionen in der Originalquelle vorhanden sind (Kapitel 40.4, Seite 332), und kann sie gegebenenfalls herunterladen (Kapitel 34.4, Seite 221).

Die durch *uscan* gewonnene Informationen werden auch auf der Seite des jeweiligen Maintainers

<https://qa.debian.org/developer.php?<Maintainer>@debian.org>
dargestellt.

Dafür benötigt *uscan* vor allem eine taugliche Datei *debian/watch*. Die Erstellung dieser Datei, die auch beim Herunterladen des Quellcodes mit *uscan* (Kapitel 34.4, Seite 221) benötigt wird, wird vom Programmskript unterstützt (Kapitel 35.4.7, Seite 239).

Wenn Dateien ausgeschlossen werden und die **.orig.tar.xz*-Datei entsprechend benannt wurde (Kapitel 10.4.1.3, Seite 32), sind entsprechende Optionen in die Datei *debian/watch* einzufügen. Diese Optionen sind *repack*, *compression*, *repacksuffix* und *dversionmangle*. Dann kann bei der nächsten neuen Upstream-Version das Herunterladen mit *uscan* durchgeführt werden. Dies gilt auch für die Prüfung mittels *uscan*, ob es eine neue Upstream-Version gibt.

Beispiel:

```
opts=repack,\
compression=xz,\
repacksuffix=+dfsg,\
dversionmangle=s/\+dfsg// \
```


12. *dh_make*

Es gibt erfreulicherweise viele nützliche Programme (Hilfsprogramme), die das Bauen von **Debian**-Paketen erleichtern und vereinheitlichen. Wer sich mit dem Bauen von **Debian**-Paketen befasst, wird immer wieder auf das Werkzeug *dh_make* stoßen. Das Paket heißt *dh-make* (mit Bindestrich).

dh_make zeigt auf, wie ein **Debian**-Paket potenziell aussehen kann. Dies ermöglicht das Erlernen des Paketierens an konkreten Beispielen. Lehrreich sind vor allem die im Programmpaket enthaltenen Vorlagen (Templates).

dh_make erstellt viele Dateien in dem Verzeichnis *debian/*. Für die meisten Pakete wird nur eine Teilmenge davon benötigt.

Die Praxis zeigt, dass jedes Mal geprüft werden muss, welche Dateien für das konkrete Paket nicht benötigt werden. Auch sind die erzeugten Dateien unvollständig und erfordern einiges an Handarbeit.

Um *dh_make* nutzen zu können, sind einige Vorbereitungen zu treffen. Dazu wird ein entsprechendes „Arbeitsverzeichnis“ angelegt. Darin wird das heruntergeladene Quellcode-Archiv abgelegt. Bevorzugt wird ein hierfür vorgesehenes Tar-Archiv.

Dieses Tar-Archiv wird dort entpackt. Dies kann auf der Konsole durchgeführt werden.mit

```
tar --extract --auto-compress --file=<Tar-Archiv>.tar.gz
```

dh_make muss in dem Verzeichnis aufgerufen werden, das den Quellcode enthält. Dabei kann es vorkommen, dass der Name des Verzeichnisses mit dem Quellcode außer Kleinbuchstaben noch weitere Zeichen enthält. *dh_make* benötigt jedoch einen Namen, der Einschränkungen unterliegt und dem Schema <paketname>-<version> entspricht. *dh_make* akzeptiert bei Paketnamen nur Kleinbuchstaben.

Damit *dh_make* die Vorlagen korrekt füllen kann, wird in solchen Fällen an *dh_make* die zusätzliche Option *--packagename* hinzugefügt. Hiermit kann die Verwendung des nachfolgenden Namens als Name des Quellcodepaketes erzwungen werden. Dieser Verzeichnisname muss die Versionsbezeichnung enthalten. Zwischen Paketnamen und Versionsbezeichnung muss ein Bindestrich stehen.

So kann nach dem Wechsel in dieses Verzeichnis dort mit

```
dh_make --createorig --packagename <SourceName>
```

das Quellarchiv in der von **Debian** gewünschten Form *<SourceName>.orig.tar.gz/xz* erzeugt werden. Daneben wird ein Entwurf der Dateien im Verzeichnis *debian/* erstellt.

Dabei werden folgende Informationen abgefragt.

Type of package = Paketklassen single, indep, library, python mit folgender Bedeutung

single Es wird ein einzelnes Binary-Paket (*.deb) erstellt. Dies ist der Normalfall

indep Es wird ein Binary-Paket erstellt, das von der Architektur unabhängig ist.

library Es werden mindestens zwei Binärdateien erstellt. Ein Bibliothekspaket, das nur die lib in */usr/lib* enthält, und ein weiteres **-dev_*.deb*-Paket, das die Dokumentation und C-Header enthält.

python

4. Mai 2025

Einzelheiten zum Paket Es werden die ermittelten Werte für *Maintainer Name*, *Email-Address*, *Date*, *Package Name*, *Version*, *License* und *Package Type* zur Bestätigung angezeigt.

Es werden folgende Dateien erzeugt. Dabei werden zunächst die Dateien gelistet, die auf jeden Fall benötigt werden, jeweils in alphabetischer Reihenfolge. Die Dateien *README.Debian* und *README.source* können zu Informationszwecke genutzt werden. Zum Schluss folgen solche Vorlagen, die nur in sehr wenigen Paketen genutzt werden.

- changelog
- control
- copyright
- rules
- salsa-ci.yml.ex
- source/format
- watch.ex
- README.Debian
- README.source
- manpage.1.ex
- manpage.sgml.ex
- manpage.xml.ex
- postinst.ex
- postrm.ex
- preinst.ex
- prerm.ex
- <packagename>.cron.d.ex
- <packagename>.doc-base.EX
- <packagename>-docs.docs

Die nicht benötigten Dateien werden aus dem Verzeichnis *debian/* entfernt.

Dass der Verzeichnisname die jeweilige Versionsbezeichnung enthalten muss, ist bei der Verwendung von *git* ungünstig. Es wird bisher davon abgesehen, dieses Hilfsprogramm im Programmskript einzusetzen.

13. Java-Pakete bauen

Das Bauen von Java-Paketen weist besondere Herausforderungen auf. Diese Besonderheiten werden im Folgenden ausführlich beschrieben.

Für das Packen von Java-Paketen gibt es eine speziellen Richtlinie, die *Debian-Java-Policy*[28].

Ferner gibt es noch eine Beschreibung über das Paketieren mit den *Javatools*[29]. Darin werden die *Java-Helper* beschrieben. Dieses Tutorial ist auch Bestandteil des *Debian-Paketes javahelper*.

Weitere Informationen erhält man auch aus der *Java-FAQ*[30], vor allem im Abschnitt 2.4¹

13.1. Herausforderungen

In einem Blog-Artikel[31] beschreibt Hans-Christoph Steiner zutreffend die Herausforderungen für ein *Debian*-Paket in Bezug auf *Java*-Software.

Das *Debian-Java*-Team muss demnach konsequent gegen die *Java*-Standardpraxis ankämpfen, alle Abhängigkeiten in einer einzigen *.jar*-Datei zu bündeln. Dies bedeutet, dass es keine gemeinsamen Sicherheitsaktualisierungen gibt. Jeder Entwickler muss jede Abhängigkeit in diesem Modell selbst aktualisieren. Das funktioniert hervorragend für große Unternehmen mit Mitarbeitern, die sich dieser Aufgabe widmen.

Für die Mehrheit der *Debian*-Anwendungsfälle funktioniert das schlecht. *Debian* hält das Versprechen ein, dass die Leute *foo* einfach passend installieren können, damit es funktioniert, und Sicherheitsaktualisierungen erhalten. Der Benutzer muss nicht einmal wissen, in welcher Sprache das Programm geschrieben ist, es funktioniert einfach.

Die Hoffnung, dass die *Java*-Entwickler-Gemeinschaft sich den Wert dieser Anwendungsfälle zu eigen macht und *Debian* hilft, indem sie es einfacher macht, *Java*-Projekte in der Standard-Distributionsmethode zu paketieren, mit gemeinsamen Abhängigkeiten, die unabhängig aktualisiert werden, hat sich bislang nur rudimentär erfüllt.

13.2. Anwendungen und Bibliotheken

Es gibt zwei Kategorien von *Java*-Paketen: Anwendungen und Bibliotheken. Sowohl *Java*-Anwendungen als auch *Java*-Bibliotheken können im Quellcode (weitere) Bibliotheken enthalten, die vorkompiliert sind.

Sollen in *Java* geschriebene Anwendungen paketiert werden, müssen in der Regel auch *Java*-Bibliotheken als *Debian*-Pakete bereitgestellt werden.

Sowohl Anwendungen als auch Bibliotheken werden in *.jar*-Dateien als *Zip*-Archiv bereitgestellt.

Beide Arten müssen grundsätzlich als *Java*-Bytecode (*.class*-Dateien, verpackt in einem *.jar*-Archiv) und mit einer „*Architecture: all*“ (Kapitel 35.4.8, Seite 244) ausgeliefert werden.

Hat man von einer zu paketierenden Anwendung oder Bibliothek zunächst nur ein (kompiliertes) *.jar*-Archiv, kann ein Blick in die Datei *MANIFEST.MF* des jeweiligen

¹<https://www.debian.org/doc/manuals/debian-java-faq/ch2.en.html#s2.4>

.jar-Archivs helfen, diejenige *URL* zu ermitteln, unter der der Quellcode veröffentlicht worden ist.

13.2.1. Java-Programme paketieren

Java-Programme sind dazu bestimmt, von Endbenutzern ausgeführt zu werden. Diese benötigen auch eine Manpage, soweit es sich um selbstständig ausführbare Programme handelt.²

Das Paket muss auch sicherstellen, dass die richtige Klasse als Hauptklasse verwendet wird.

Zusätzliche Klassen im Paket müssen in ein oder mehrere Java-Archiv(e) (**.jar*-Datei(en)) gepackt werden, die in */usr/share/java* (wenn sie für die Verwendung durch andere Programme vorgesehen sind) oder in */usr/share/<package>* als ein „privates“ Verzeichnis abgelegt werden können.

13.2.2. Java-Bibliotheken paketieren

Java-Bibliotheken dienen der Erfüllung von Abhängigkeiten von Programmen. Dies können Build- und/oder Laufzeitabhängigkeiten sein.

13.2.3. Name des Java-Paketes

Eine besondere Herausforderung ist die Bildung des korrekten Namen des Java-Paketes.

13.3. Abhängigkeiten bei Java-Paketen

Java-Anwendungen hängen immer auch von Java-Bibliotheken ab. Außerdem wird für die Java-Pakete immer *default-jdk* als Abhängigkeit benötigt. Dahinter verbirgt sich in der Regel die aktuell verfügbare OpenJDK-Version, eine freie Java Implementierung mit Langzeitunterstützung (JDK = Java Development Kit).

Daneben gibt es spezielle Abhängigkeiten, die sich aus dem Build-System ergeben. Diese werden bei den einzelnen Build-Systemen im Folgenden besprochen.

13.3.1. Weitere Abhängigkeiten feststellen

Java-Entwickler fügen ihren Quellcode-Paketen die benötigten Build-Abhängigkeiten - auch Bibliotheken von Drittanbietern - in kompilierter Form als **.jar*-Archive hinzu.

Gerade unter Java-Entwicklern ist es leider sehr verbreitet, auf irgendwo im **World Wide Web** veröffentlichte Bibliotheken zurückzugreifen und diese dann vorkompiliert mit auszuliefern. Diese Bibliotheken müssen durch in **Debian** veröffentlichte Pakete ersetzt werden.

Zur Feststellung von Abhängigkeiten ist also im Quellcode nach **.jar*-Archiven zu suchen.

In diesem Falle ist für den **Debian**-Maintainer ein Repacking zum Entfernen dieser Dateien aus dem Upstream-Tarball notwendig. (Kapitel 36, Seite 253)

Die ausgeschlossenen Bibliotheken müssen durch eigenständige Pakete verfügbar gemacht werden.

²Kapitel 2.3 der *Java-Policy*[28]

13.3.2. Abhängigkeiten ermitteln

Hat man Abhängigkeiten festgestellt, so ist zu ermitteln, wie diese Abhängigkeiten erfüllt werden können. Hierzu ist als erstes zu prüfen, ob bereits entsprechende **Debian**-Pakete existieren. Um zu ermitteln, ob eine Abhängigkeit bereits in **Debian** paketierte wurde, gibt es mehrere Wege.

Debian-Wiki (für Maven-Pakete):

<https://wiki.debian.org/Java/MavenPkgs>

13.4. Build-Systeme für Java-Pakete

Zum Bauen von **Java**-Paketen werden verschiedene Build-Systeme genutzt. Hierbei handelt es sich um *maven*, *ant* und *gradle*. Daraus ergeben sich Eigenheiten, die beim Paketieren für **Debian** beachtet werden müssen. Es gibt auch **Java**-Pakete, die ohne eines dieser Build-Systeme gebaut werden. Die folgenden Beschreibungen orientieren sich an den gemachten Erfahrungen der Autoren.

13.4.1. Das Build-System *maven*

Apache Maven ist ein Werkzeug zur Verwaltung und zum Verstehen von Softwareprojekten.

Ein wesentliches Erkennungsmerkmal ist die Datei *pom.xml* für das **Maven**-Build-System. Diese enthält alle Informationen zum jeweiligen Softwareprojekt und folgt einem standardisierten *XML*-Format.

Wesentlich ist auch die Standard-Verzeichnisstruktur eines **Maven**-Projektes, die im Folgenden dargestellt wird.[32]

src/main/java Anwendungs- / Bibliotheksquellen
src/main/resources Anwendungs- / Bibliotheksressourcen
src/main/filters Ressourcenfilterdateien
src/main/webapp Webanwendungsquellen
src/test/java Testquellen
src/test/resources Ressourcen testen
src/test/filters Testen Sie Ressourcenfilterdateien
src/it Integrationstests (hauptsächlich für Plugins)
src/assembly Assembly-Deskriptoren
src/site Website

LICENSE.txt Lizenz des Projektes

NOTICE.txt Hinweise und Zuordnungen für Bibliotheken, von denen das Projekt abhängt

README.txt Readme des Projektes

pom.xml Beschreibung des Projektes und der Konfiguration

In den meisten Fällen reicht es aus, den Zweig *src/main/* als **Debian**-Paket zu bauen. Dies gilt besonders dann, wenn Bibliotheken gebaut werden, die als Abhängigkeiten für Anwendungen benötigt werden.

Das Upstream-Paket muss bestimmte Voraussetzungen erfüllen, um mit dem Build-System **Maven** gebaut werden zu können.

Dazu gehört auf jeden Fall mindestens eine *pom.xml*-Datei. Diese Datei kann an verschiedenen Stellen innerhalb des Quellcodes liegen. Manchmal wird sie - besonders im **Maven**-Repository (<https://mvnrepository.com/>) - nur zusätzlich bereitgestellt.

Wenn vorhanden, ist es jedoch oft besser, **GitHub** oder andere **Git**-Repositoryen als Quelle zu wählen. Dabei ist auf die Version zu achten! Für manche Pakete wird der

Quellcode auch unter der Domain <https://gitbox.apache.org/repos/asf> veröffentlicht.

13.4.2. Paketieren mit *maven*

Für *maven*-basierte Pakete wird die Verwendung von *maven-debian-helper*³ empfohlen.

In der Datei *debian/rules* (Kapitel 47.5, Seite 398) wird dazu `--buildsystem=maven` hinzugefügt.

Zur Unterstützung des Bauens mit *maven* enthält dieses Paket folgende Befehle:

mh_genrules(1) generiert, zumindest teilweise, die Datei *debian/rules*

mh_lspoms(1) sucht nach allen POM-Dateien, die im Quellcode des Projektes angegeben sind.

mh_make(1) generiert durch Lesen der Informationen aus der Maven-POM das Debian-Paket.

mh_resolve_dependencies(1) löst die Abhängigkeiten auf und erzeugt die Datei `<Paketname>.substvars`, die die Liste der abhängigen Pakete enthält, zur Nutzung von *debian/control*.

Zu diesen Befehlen gibt es auch entsprechende *Man-Pages*.

Hiervon wird zunächst nur der Befehl *mh_make*⁴ im Plugin *build-gbp-maven-plugin.sh* genutzt. (Kapitel 47, Seite 385)

Bei der Nutzung von *mh_make* ist zu beachten, dass dort, wo *mh_make* gestartet wird, alle Abhängigkeit des zu bauenden Paketes vorhanden sein sollten. Andernfalls sind etwaige Einträge in den einschlägigen Dateien (z.B. *debian/control*) manuell nachzuholen. (Kapitel 47.4, Seite 394)

Dazu empfiehlt es sich, diese Operationen in einer eigens dafür eingerichteten *Chroot* durchzuführen, in der alle Abhängigkeiten installiert werden können, ohne das eigentliche Host-System zu belasten. Die Einrichtung derselben ist in Kapitel 19.5 (Seite 69) beschrieben.

Von *mh_make* werden folgende Dateien im Verzeichnis *debian/* angelegt:

- *maven.cleanIgnoreRules*
- *maven.rules*
- *maven.ignoreRules*
- *maven.properties*
- `<Paketname>.poms`
- *maven.publishedRules*

Zusätzlich werden von *mh_make* auch die (Standard-)Dateien

- *changelog*
- *control*
- *copyright*
- *rules*
- *README.source*

erstellt. Dies ist bei den weiteren Arbeiten an diesen Dateien zu berücksichtigen (Kapitel 35.4, Seite 229).

Mit *apt-file find*⁵ kann festgestellt werden, ob es zu einer *artifactID* ein Debian-Paket gibt. Diese Funktion wird auch von *mh_make* genutzt. Die *artifactID* befindet sich in der *pom.xml*-Datei aus dem Upstream-Code.

³<https://manpages.debian.org/unstable/maven-debian-helper/index.html>

⁴[urlhttps://manpages.debian.org/unstable/maven-debian-helper/mh_make.1.en.html](https://manpages.debian.org/unstable/maven-debian-helper/mh_make.1.en.html)

⁵<https://manpages.debian.org/unstable/apt-file/apt-file.1.en.html>

Unter wiki.debian.org/Java/MavenPkgs/Unstable befindet sich eine Liste aller in Debian bereits paketierte *Maven*-Pakete.⁶

`mh_make` erstellt aus der `pom.xml` auch die Datei `debian/<Paketname>.poms`.

In dieser Datei werden alle im Paket vorkommende `pom.xml`-Dateien mit ihrem relativen Pfad aufgelistet. Diese können dann dort mit den aufgelisteten Optionen versehen werden.

```
43 <debian/JavaPackage.poms 43>≡
# List of POM files for the package
# Format of this file is:
# <path to pom file> [option]*
# where option can be:
# --ignore: ignore this POM and its artifact if any
# --ignore-pom: don't install the POM. To use on POM files that are created
# temporarily for certain artifacts such as Javadoc jars.
# [mh_install, mh_installpoms]
# --no-parent: remove the <parent> tag from the POM
# --package=<package>: an alternative package to use when installing
# this POM and its artifact
# --has-package-version: to indicate that the original version of the
# POM is the same as the upstream part of the version for the package.
# --keep-elements=<elem1,elem2>: a list of XML elements to keep in the POM
# during a clean operation with mh_cleanpom or mh_installpom
# --artifact=<path>: path to the build artifact associated with this POM,
# it will be installed when using the command mh_install. [mh_install]
# --java-lib: install the jar into /usr/share/java to comply with Debian
# packaging guidelines
# --usj-name=<name>: name to use when installing the library in
# /usr/share/java
# --usj-version=<version>: version to use when installing the library
# in /usr/share/java
# --no-usj-versionless: don't install the versionless link in
# /usr/share/java
# --dest-jar=<path>: the destination for the real jar.
# It will be installed with mh_install. [mh_install]
# --classifier=<classifier>: Optional, the classifier for the jar.
# Empty by default.
# --site-xml=<location>: Optional, the location for site.xml if it
# needs to be installed. Empty by default. [mh_install]
#
```

⁶Dank an Thorsten Glaser und der Firma Tarent <https://www.tarent.de/> für diese Unterstützung

Eine häufig verwendete Zeile ist `pom.xml -no-parent -has-package-version`. Dies bedeutet, dass das `<parent>` Tag beim Bauen aus der POM entfernt wird. Die Option `-has-package-version` gibt an, dass die Originalversion der POM dieselbe sei, wie der Upstream-Teil der Paketversion.

Diese Datei kann innerhalb des Bauprozesses einer neuen Debian-Revision angepasst werden. (Kapitel 47.4, Seite 394).

Liegt die `pom.xml` im Wurzelverzeichnis des Quellcodes, werden durch `mh_make` die weiteren Dateien für das Verzeichnis `debian/` erstellt.

Im Maven-Repository wird die Datei `pom.xml` oft nur separat bereitgestellt. Diese Datei wird dann im `debian/` Verzeichnis abgelegt.

Daher sind hier einige Einträge in `debian/rules` (Kapitel 35.4.8, Seite 244 erforderlich:

Nach dem Aufruf aller `debhelper` (`dh $@`) wird das `dh_auto_build` zunächst mit einem `override` außer Kraft gesetzt, um noch einige vorbereitenden Konfigurationen durchzuführen.

Dazu gehört auch das Kopieren von Dateien an den Ort, wo das Build-System diese Dateien benötigt, um das gewünschte Programm zu bauen.

```
44 <maven-build 44>≡
    override_dh_auto_build:
        cp debian/pom.xml .
        dh_auto_build
```


Weitere Literatur

1. <https://wiki.debian.org/Java/MavenBuilder>
2. <https://wiki.debian.org/Java/MavenRepoSpec>
3. <https://wiki.ubuntu.com/JavaTeam/Specs/MavenSupportSpec>

Mit `apt-file find`⁷ kann festgestellt werden, ob es zu einer `artifactID` ein Debian-Paket gibt. Diese Funktion wird auch von `mh_make` genutzt. Die `artifactID` befindet sich in der `pom.xml`-Datei aus dem Upstream-Code.

Unter wiki.debian.org/Java/MavenPkgs/Unstable befindet sich eine Liste aller in Debian bereits paketierten *Maven*-Pakete.⁸

13.4.3. Paketieren mit *ant*

Das Build-System *ant* erkennt man daran, dass es eine Datei `build.xml` gibt.⁹

13.4.4. Paketieren mit *gradle*

13.5. Java-Pakete bauen ohne Build-System

Es ist auch möglich, einfach eine Java-Bibliothek aus einem Verzeichnis mit *.java-Dateien zu bauen. Dies ist dann sinnvoll, wenn aus einem größeren Programmpaket nur eine oder wenige einzelne Java-Klasse benötigt wird, um Buildabhängigkeiten anderer Pakete zu erfüllen.

Dabei gibt es für ein solches Java-Verzeichnis kein Build-System wie die noch zu beschreibenden *ant* (Kapitel 13.4.3, Seite 45 bzw. *maven* (Kapitel 13.4.1, Seite 41).

45

```

<java-builds 45>≡
export DH_VERBOSE=1
export DH_OPTIONS=-v

Class-Path: src/net/numericalchameleon/util/phoneticalphabets.jar
export CLASSPATH=/usr/share/java/sugar.jar

%:
    dh $@ --with javahelper --sourcedirectory=src/net/numericalchameleon/util/

override_jh_build:

    javac -encoding UTF-8 src/net/numericalchameleon/util/phoneticalphabets/*
    jh_build
    jar cvf src/phoneticalphabets.jar \
    src/net/numericalchameleon/util/phoneticalphabets/*.class

    javac -encoding UTF-8 src/net/numericalchameleon/util/spokennumbers/*
    jh_build

```

⁷<https://manpages.debian.org/unstable/apt-file/apt-file.1.en.html>

⁸Dank an Thorsten Glaser und der Firma Tarent <https://www.tarent.de/> für diese Unterstützung

⁹<https://wiki.debian.org/Java/Packaging/Ant>

4. Mai 2025

Das *javahelper*-Paket bietet Hilfestellung beim Bauen mit *dh*. Diese wird für die Java-Paketierung dringend empfohlen.

Ferner wird die Datei *debian/javabuild* benötigt. Diese enthält zwei Spalten.

Anzugeben ist in der ersten Spalte die zu bauende **.jar*-Datei und in der zweiten Spalte das Verzeichnis des Quellcodes, in dem die **.java*-Dateien liegen.

Beispiel:

```
#NameOfJarFile SourceDirToPackage
```

Das Skript ermöglicht das Erstellen dieser Datei (Kapitel 35.4.10, Seite 247).

14. Mozilla-Erweiterungen bauen

Das Paketieren von Mozilla-Erweiterungen als Debian-Pakete bietet für Nutzer und Administratoren den Vorteil, dass Erweiterungen zusammen mit der Hauptanwendung aktualisiert werden können.

Debian-Pakete werden zentral für alle Systemnutzer installiert. Die Aktualisierungen erfolgen auch mit dem Paketmanagementsystem der Distribution. Es bedarf dann nur eines Werkzeuges für diesen Zweck.

Bei Sicherheitsaktualisierungen oder neuen Versionen der Anwendungen sind diese Aspekte besonders relevant.

Dies hat den weiteren Vorteil, dass alle Nutzer mit den gleichen Versionsständen arbeiten.

14.1. Quellen der Erweiterungen

Unter `addons.mozilla.org` findet man Erweiterungen für den Firefox veröffentlicht von Mozilla.

Für den Thunderbird werden unter `addons.thunderbird.net` die Erweiterungen veröffentlicht.

Diese liegen dort als *.xpi*-Archive vor. Sofern dies Quellcode-Pakete sind, gibt es nicht zwingend auch separate Quellcode-Archive.

Im Falle einer *.xpi*-Datei bedarf es spezieller Einträge in der Datei *debian/watch* zum Umpaketieren. Das *.xpi*-Format ist ein besonders spezifiziertes *.zip*-Archiv. Nachstehend ist die *debian/watch*-Datei für die Thunderbird-Erweiterung Mailmindr dargestellt.

```
47 <watch4xpi 47>≡
    version=4
    opts=\
    repack,compression=xz,\
    uversionmangle=s/-?([^\d.]+)/~$1/;tr/A-Z/a-z/, \
    filenamemangle=s/.\+\/v?(\d\S+)\.*/$1/,
    https://addons.thunderbird.net/en-US/thunderbird/addon/mailmindr/versions/ \
    (\d.\d.\d)
```

4. Mai 2025

Manchmal findet man auch Quellcode-Veröffentlichungen von Personen und Projekten beispielsweise auf Github (github.com) oder auf selbstgehosteten Seiten. Diese können auch als *.tar.gz*- oder *.zip*-Archive vorliegen.

Erfahrungsgemäß sind dort die Versionen, die bereits im *addons*-Archiv veröffentlicht worden sind, nicht immer zeitnah als neue Version gekennzeichnet.

14.2. Integration ins Dateisystem

Um alle gewünschten **Mozilla**-Erweiterungen einheitlich zu bauen, werden die benötigten Teile des Skriptes als *Webext-Plugin* realisiert (Kapitel 48, Seite 399). Damit wird auch eine einheitliche Integration in das System gewährleistet.

Ab der Version (≥ 78.2) des **Thunderbirds** können ausschließlich *Mail=Extensions* genutzt werden. Die alten (für TB $\leq 68.x$) Erweiterungen funktionieren nicht mehr.

Die Erweiterung muss nun im Verzeichnis */usr/share/webext* mit der Bezeichnung (id) aus der Datei *manifest.json* ergänzt und mit der Endung (.xpi) als Zip-Archiv abgelegt werden (Kapitel 48.2.4, Seite 403).

Dazu wird die Datei *debian/rules* entsprechend erweitert (Kapitel 48.2.2, Seite 401). Zusätzlich sind immer die Dateien *<Paketname>.install* (Kapitel 48.2.4, Seite 403) und *<Paketname>.links* (Kapitel 48.2.6, Seite 403) so zu erstellen, dass **Thunderbird** die Erweiterung auch findet.

15. Python-Pakete bauen

16. Dokumentation paketieren

16.1. Dokumentation für Debian bauen

Diese Dokumentation kann als Beispiel dafür dienen, wie man Dokumentationen als Debian-Paket bereitstellt.

Diese Dokumentation ist bereits auf *salsa.debian.org*¹ veröffentlicht. Dies bietet die Möglichkeit, das Buch als **natives** Debian-Paket zu bauen.

Ein natives Quellpaket ist nach der Debian-Policy² ein Paket, das nicht zwischen Debian-Paketveröffentlichungen und Upstream-Veröffentlichungen unterscheidet. Ein natives Quellpaket enthält eine einzelne *tar*-Datei mit Quellmaterial, und die Versionierung hat keine Debian-spezifische Komponente. Native Pakete werden normalerweise (aber nicht ausschließlich) für Software verwendet, die keine unabhängige Existenz außerhalb von Debian hat, wie z. B. Software, die speziell für ein Debian-Paket geschrieben wurde. Gleiches gilt für Dokumentation, die speziell für Debian geschrieben wurde. Ein nicht-natives Quellpaket trennt die Upstream-Veröffentlichung von der Debian-Paketierung und allen Debian-spezifischen Änderungen.

Die Information, dass es sich um ein natives Paket handelt, wird in der Datei *debian/source/format* als *3.0 (native)* abgelegt (Kapitel 35.4.2, Seite 231).

Native Pakete benötigen keine Datei *debian/upstream/metadata* (Kapitel 35.4.4, Seite 233). und keine Datei *debian/watch* (Kapitel 35.4.7, Seite 239). Die Versionsbezeichnung enthält keine Revisionsbezeichnung (Kapitel 37.1.1, Seite 293 und Kapitel 37.5.3, Seite 316).

Das Bauen *nativer* Pakete wird vom Programmskript unterstützt (Kapitel 37.5.3, Seite 316).

16.2. Upstream-Dokumentation bauen

Wird vom Upstream-Projekt eine umfangreiche oder mehrsprachige Dokumentation bereitgestellt, empfiehlt es sich, diese Dokumentation in gesonderten Debian-Paketen zu veröffentlichen.

¹<https://salsa.debian.org/ddp-team/dpb>

²Kapitel 4 der Debian-Policy[7]

17. Metapakete bauen

Metapakete sind solche Debian-Pakete, die von einer Gruppe von Paketen abhängen. Damit werden Gruppen von Paketen zusammengefasst, bei denen es sinnvoll ist, sie gemeinsam zu installieren. Beispiele sind Erweiterungen für eine bestimmte Software.

Metapakete weisen einige Besonderheiten auf.

17.1. Kein Upstream-Quellcode

Da es keinen Upstream-Quellcode gibt, benötigt ein Metapaket auch keine Datei *debian/watch*

17.2. Natives Debian-Paket

17.2.1. *debian/source/format*

17.2.2. *debian/control*

17.2.3. *debian/rules*

17.2.4. *debian/changelog*

18. Konfiguration zur Installation

18.1. *debconf*

Debconf ist ein Konfigurationsmanagementsystem für *Debian*. Dies dient dazu, Informationen für die Installation vom Nutzer abzufragen und diese dann bei der Installation zu berücksichtigen.¹

Mit `sudo dpkg-reconfigure debconf` kann *debconf* selbst konfiguriert werden. Dies beinhaltet auch die grafische Oberfläche.

```
first step: https://wiki.debian.org/debconf  
           https://wiki.debian.org/debconf
```

18.2. *dbconfig-common*

Dbconfig-common stellt eine einfache, zuverlässige und konsistente Methode zur Verwaltung von Datenbanken bereit, die von *Debian*-Paketen verwendet werden.

¹<https://wiki.debian.org/debconf>

19. System einrichten

Es wird davon ausgegangen, dass bereits ein installiertes und lauffähiges **Debian**-System existiert. Dabei ist es unerheblich, welcher Zweig (*unstable*, *testing* oder *stable*) darauf läuft.

Zum Signieren der vom Programmskripts erzeugten Dateien muss ein *GPG*-Schlüssel des Nutzers auf dem System zur Verfügung stehen. Dieser sollte sinnvollerweise der für das **Debian**-Repositorium vorgesehene Schlüssel sein.

Dies gilt auch, wenn für das Bauen eine virtuelle Maschine eingerichtet und genutzt wird.

19.1. Abhängigkeiten für das Programmskript

Die Pakete, die zum Einsatz des Programmskriptes erforderlich sind, sind in den Kopfzeilen desselben aufgeführt (Kapitel 45.3, Seite 379).

Einige der darin enthaltenen Programme werden im folgenden beschrieben:

19.1.1. Generelle Abhängigkeiten

Die nachstehend aufgeführten, hilfreichen Programme werden regelmäßig vom Programmskript zum Bauen eines **Debian**-Paketes eingesetzt:

mk-origtargz aus dem Paket *devsripts* benennt den Tarball der Originalautoren um, ändert wahlweise die Komprimierung und entfernt unerwünschte Dateien. Dazu stehen unterschiedliche Optionen zur Verfügung[33] (s. Verwendung in Kapitel 34.3.6, Seite 205).

gbp import-orig aus dem Paket *git-buildpackage* importiert eine neue Upstream-Version in ein **Git**-Repositorium (s. Verwendung in Kapitel 34.3.11, Seite 218).

gbp import-dsc aus dem Paket *git-buildpackage* importiert ein existierendes **Debian**-Quellpaket in ein **Git**-Repositorium (s. Verwendung in Kapitel 32.6, Seite 156). Auf diesem Weg kann ein Mentor ein Paket „sponsorn“ (Kapitel 32.7, Seite 158).

dh_make aus dem Paket *dh-make* kann die benötigten **Debian**-Dateien erstellen. Die Praxis zeigt, dass dies sehr unvollständig ist und einiges an Handarbeit erfordert. Daher wird bisher davon abgesehen, dieses Hilfsprogramm im Programmskript einzusetzen. (s. Kapitel 12, Seite 37)

debmake -cc aus dem gleichnamigen Paket *debmake* durchsucht die Quelldateien nach Copyright- und Lizenztexten (s. Kapitel 10.1.1, Seite 27). Dieser Befehl wird vom Programmskript zur Erstellung der Datei *debian/copyright* verwandt. (s. Verwendung in Kapitel 35.4.5, Seite 234).

gbp dch ist auch im Paket *git-buildpackage* enthalten. Im Programmskript wird *dch* über *gbp dch* verwandt (s. Verwendung in Kapitel 37.1, Seite 291). Weitere Optionen für *dch* können an *gbp dch* durch *--dch-opt=<dch-Optionen>* übergeben werden.

gbp buildpackage ist ebenfalls im Paket *git-buildpackage* enthalten. Mit diesem Befehl wird ein Paket aus dem **Git**-Repositorium erstellt (s. Verwendung in Kapitel 37.5.4, Seite 320)

lintian aus dem Paket *lintian* ist ein Analysewerkzeug für **Debian**-Pakete. Es berichtet Fehler und Verletzungen der **Debian**-Policy [7]. Jeder **Debian**-Betreuer sollte

4. Mai 2025

mit diesem Werkzeug das Paket vor dem Hochladen ins Archiv prüfen. Das Programmskript erwendet *lintian* mit den Optionen *--pedantic* und *--display-experimental*. (s. Verwendung in Kapitel 40.3, Seite 330).

Während des Bauens wird *lintian* durch *dh_lintian* aus dem Paket *devscripts* aufgerufen.

uscan in *devscripts*

debsign in *devscripts* auch zum Signieren beim Sponsoring

dput in *dput* bzw *dput-ng*

mh_make (Plugin)

19.1.2. Abhängigkeiten für das Bauen von Java-Paketen

Die folgenden Abhängigkeiten werden für das Bauen mit Build-Systemen für Java-Pakete benötigt.

```
gradle-debian-helper, maven-debian-helper, libmaven-bundle-plugin-java
```

19.1.3. Abhängigkeiten für Erweiterungen, die Zip-Archive sind

Manche Erweiterungen (Plugins) werden als *Zip-Archive* (z. B. für Libreoffice *.oxt*, für Mozilla *.xpi*) verteilt, die den Quellcode enthalten.

Für diese Erweiterungen muss in der Datei *debian/control* (Kapitel 35.4.6, Seite 235) *zip* als Buildabhängigkeit aufgeführt werden.

Für Mozilla-Erweiterungen wird dies vom *Web-Extension-Plugin* erledigt (Kapitel 48.2.3, Seite 402).

19.2. Verzeichnisse und Dateien

19.2.1. Pfade zu den Projekten

Es ist sinnvoll, die Paketierprojekte jeweils als Unterverzeichnisse eines Projektverzeichnisses anzulegen. Ein Beispiel für ein solches Projektverzeichnis ist:

```
~/Projekte/Git/01_Salsa
```

Dieser Pfad wird in jeder Konfigurationsdatei in der Variablen *ProjectPath* hinterlegt.

Ferner kann dieser Pfad auch in der Variablen *DefaultProjectPath* für viele Projekte als Standardwert in der Datei *~/debian_project/DefaultValues* aufgenommen werden (Kapitel 19.2.2.2, Seite 60).

PrjPath ist das Unterverzeichnis des Projektverzeichnisses (*ProjectPath*), in dem das einzelne Projekt liegt. Der Name dieses Unterverzeichnisses ist der Name des entsprechenden Projektes (*OrigName*). In dieses Unterverzeichnis werden die fertigen Pakete abgelegt.

In diesem Unterverzeichnis befindet sich ein weiteres Unterverzeichnis (*GitPath*), welches das Git-Repository enthält.

Diese Unterverzeichnisse werden, wenn nötig, vom Programmskript angelegt (s. a. Kapitel 32.2.1, Seite 129)

19.2.2. Konfigurationsdateien

19.2.2.1. Für jedes Projekt

Für jedes Projekt wird eine eigene Konfigurationsdatei erstellt. Diese wird im Homeverzeichnis des Nutzers im Verzeichnis `.debian_project/` als Datei `<Projektname>` abgelegt. Darin werden projektspezifische Informationen hinterlegt.

Die Konfigurationsdatei ist technisch ein Shell-Skript, welches von dem Programmskript erstellt und geladen wird. Dieses Shell-Skript enthält Variablen, denen dort Werte zugewiesen werden. Die in der Konfigurationsdatei mit Werten versehenen Variablen können durch das Laden vom Programmskript verwendet werden.

Ferner enthält die Datei Kommentare. Diese können vom Nutzer beliebig erstellt werden. Ein Teil der Kommentare wird jedoch vom Programmskript erstellt. Diese enthalten ein Eigenschafts-Werte-Paar, welches wie die Zuweisung eines Wertes an eine Variable aufgebaut ist. Technischer Hintergrund ist, dass im Eigenschaftsnamen sinnvollerweise Zeichen vorkommen, die im Bezeichner einer Variablen nicht verwendet werden dürfen.

Existiert die Konfigurationsdatei, so wird sie zunächst geladen. (Kapitel 33.1, Seite 165). Man kann sie dann, wenn nötig, editieren. Andernfalls wird diese Datei durch das Skript erstellt. (Kapitel 32.1, Seite 111)

Folgende Informationen werden dort hinterlegt:

```
# !/usr/bin/bash
# ConfigFile for <OrigName>
## General parameters
SourceName
PackageName
ProjectPath = ~/Projekte/Git/01_Salsa
SalsaName
Java-Package
    SalsaName = java-team/<SourceName>.git
    JavaFlag
    MavenPluginFlag
    Maintainer =Maintainer=Debian_Java_Maintainers
        _@lt@pkg-java-maintainers@lists.alioth.debian.org@gt@
    Uploader =Mechtilde_Stehmann_@lt@mechtilde@debian.org@gt@
Web-Extension-Packages
    SalsaName = webext-team/
    WebextFlag
    Maintainer
    Uploader
Python-Packages SalsaName = python-team/packages/
    PythonFlag
    Maintainer
    Uploader
DefaultBranch
RecentBranch = debian/sid
RecentUpstreamSuffix = .tar.gz
RecentRepackSuffix = +dfsg | +ds
master_Dist
DownloadUrl
DownloadZip
Maintainer Mechtilde Stehmann <mechtilde@debian.org>
ExcludeFile
```

4. Mai 2025

19.2.2.2. Für viele Projekte

In der Datei `~/.debian_project/DefaultValues` werden Variablen Werte zugewiesen, die für viele Projekte gelten.

```
60 <DefaultValues 60>≡
    #!/usr/bin/bash

    HOME=/home/<user>
    DefaultProjectPath=${HOME}/Projekte/Git/01_Salsa
```

19.2.2.3. Fingerprint des Maintainer-Schlüssels

Im Verzeichnis `~/.debian_project/` befindet sich auch eine Datei *fingerprint*, die den Fingerprint des Maintainer-Schlüssels enthält. Mit diesem Schlüssel werden die Pakete signiert. (Kapitel 32.9, Seite 161)

19.2.3. .bashrc

In die Datei *.bashrc* sind folgende Einträge aufzunehmen:

```
DEBFULLNAME="<Vor- und Nachname des Maintainer>"
DEBEMAIL="<E-Mail-Adresse des Maintainers>"
export DEBEMAIL DEBFULLNAME
```

Verschiedene Debian-Werkzeuge erkennen Ihre E-Mail-Adresse und Ihren Namen anhand der Shell-Umgebungsvariablen `$DEBEMAIL` und `$DEBFULLNAME`.¹

Diese Einträge werden auch von der Funktion *DEBValues* des Programmskripts ausgelesen. (Kapitel 32.4.2, Seite 136)

Auch für *dquilt* ist die Datei *.bashrc* zu ergänzen. (Kapitel 19.6, Seite 69)

19.3. PBuilder einrichten

Gbp buildpackage verwendet *cowbuilder*. Der Befehl *cowbuilder* ist ein Wrapper für *pbuilder*, welcher die Nutzung eines *pbuilder*-ähnlichen Interfaces in einer *cowdancer*-Umgebung ermöglicht.

19.3.1. Chroot

Das Paket *pbuilder* enthält Programme, um eine *Chroot*-Umgebung aufzubauen und zu betreiben.

Chroot bedeutet *Change root*

Es entspricht guter Praxis, **Debian**-Pakete in einer *Chroot* zu bauen. Damit soll gewährleistet werden, dass das Paket mit den Ressourcen der jeweilige Distribution gebaut werden kann.

Beim Bauen eines **Debian**-Paketes in dieser *Chroot*-Umgebung wird nämlich geprüft, ob alle erforderlichen Build-Abhängigkeiten erfüllbar sind. Hierdurch können FTBFS-Fehler (FTBFS = Failed To Build From Source) vermieden werden. Unter Umständen kann es jedoch noch zu solchen Fehlermeldungen kommen.

¹Kapitel 3.1 in [11]

19.3.2. Konfiguration des Pbuilders

Zunächst muss das Verzeichnis `/var/cache/pbuilder/result` angelegt werden. Dieses Verzeichnis muss für den Nutzer beschreibbar sein. Danach wird die Konfiguration des `pbuilder` durchgeführt.

Die Konfiguration des `pbuilder` ist nicht völlig trivial, was gegebenenfalls die Fehlersuche erschwert. Daher gehen wir hier sehr ausführlich darauf ein.

Die Standard-Konfiguration wird aus der Datei `/usr/share/pbuilder/pbuilderrc` genommen.

```
# pbuilder defaults; edit /etc/pbuilderrc to override these and see
# pbuilderrc.5 for documentation

# Set how much output you want from pbuilder, valid values are
# E => errors only
# W => errors and warnings
# I => errors, warnings and informational
# D => all of the above and debug messages
LOGLEVEL=I
# if positive, some log messages (errors, warnings, debugs) will be colored
# auto => try automatically detection
# yes => always use colors
# no => never use colors
USECOLORS=auto

BASETGZ=/var/cache/pbuilder/base.tgz
#EXTRAPACKAGES=""
#export DEBIAN_BUILDARCH=athlon
BUILDPLACE=/var/cache/pbuilder/build
# directory inside the chroot where the build happens. See #789404
BUILDDIR=/build
# what be used as value for HOME during builds. See #441052
# The default value prevents builds to write on HOME, which is prevented on
# Debian builds too. You can set it to $BUILDDIR to get a working HOME, if
# you need to.
BUILD_HOME=/nonexistent
MIRRORSITE=http://deb.debian.org/debian
#OTHERMIRROR="deb http://www.home.com/updates/ ./"
#export http_proxy=http://your-proxy:8080/
USESHM=yes
USEPROC=yes
USEDEVFS=no
USEDEVPTS=yes
USESYSFS=yes
USENETWORK=no
USECGROUP=yes
BUILDRESULT=/var/cache/pbuilder/result/

# specifying the distribution forces the distribution on "pbuilder update"
#DISTRIBUTION=sid
# specifying the architecture passes --arch= to debootstrap; the default is
# to use the architecture of the host
#ARCHITECTURE=$(dpkg --print-architecture)
# specifying the components of the distribution, for instance to enable all
# components on Debian use "main contrib non-free" and on Ubuntu "main
# restricted universe multiverse"
COMPONENTS="main"
#specify the cache for APT
APTCACHE="/var/cache/pbuilder/aptcache/"
APTCACHEHARDLINK="yes"
REMOVEPACKAGES=""
#HOOKDIR="/usr/lib/pbuilder/hooks"
HOOKDIR=""
EATMYDATA=no
# NB: this var is private to pbuilder; ccache uses "CCACHE_DIR" instead
# CCACHEDIR="/var/cache/pbuilder/ccache"
CCACHEDIR=""

# make debconf not interact with user
export DEBIAN_FRONTEND="noninteractive"

#for pbuilder debuild
BUILDSOURCEROOTCMD="fakeroot"
PBUILDERROOTCMD="sudo -E"
```

4. Mai 2025

```
# use cowbuilder for pdebuild
#PDEBUILD_PBUILDER="cowbuilder"

# Whether to generate an additional .changes file for a source-only upload,
# whilst still producing a full .changes file for any binary packages built.
SOURCE_ONLY_CHANGES=no

# additional build results to copy out of the package build area
#ADDITIONAL_BUILDEXTS=(xunit.xml .coverage)

# command to satisfy build-dependencies; the default is an internal shell
# implementation which is relatively slow; there are two alternate
# implementations, the "experimental" implementation,
# "pbuilder-satisfydepends-experimental", which might be useful to pull
# packages from experimental or from repositories with a low APT Pin Priority,
# and the "aptitude" implementation, which will resolve build-dependencies and
# build-conflicts with aptitude which helps dealing with complex cases but does
# not support unsigned APT repositories
PBUILDERSATISFYDEPENDSCMD="/usr/lib/pbuilder/pbuilder-satisfydepends"

# Arguments for $PBUILDERSATISFYDEPENDSCMD.
# PBUILDERSATISFYDEPENDSOPT=()

# You can optionally make pbuilder accept untrusted repositories by setting
# this option to yes, but this may allow remote attackers to compromise the
# system. Better set a valid key for the signed (local) repository with
# $APTKEYRINGS (see below).
ALLOWUNTRUSTED=no

# Option to pass to apt-get always.
export APTGETOPT=()
# Option to pass to aptitude always.
export APTITUDEOPT=()

# Whether to use debdelta or not. If "yes" debdelta will be installed in the
# chroot
DEBDELTA=no

#Command-line option passed on to dpkg-buildpackage.
#DEBBUILD_OPTS="-IXXX -iXXX"
DEBBUILD_OPTS=""

#APT configuration files directory
APTCONFDIR=""

# the username and ID used by pbuilder, inside chroot. Needs fakeroot, really
BUILDUSERID=1234
BUILDUSERNAME=pbuilder

# BINDMOUNTS is a space separated list of things to mount
# inside the chroot.
BINDMOUNTS=""

# Set the debootstrap variant to 'buldd' type.
DEBOOTSTRAP_OPTS=(
  '--variant=buldd'
  '--force-check-gpg'
)
# or unset it to make it not a buldd type.
# unset DEBOOTSTRAP_OPTS

# Keyrings to use for package verification with apt, not used for debootstrap
# (use DEBOOTSTRAP_OPTS). By default the debian-archive-keyring package inside
# the chroot is used.
APTKEYRINGS=()

# Set the PATH I am going to use inside pbuilder: default is
# "/usr/sbin:/usr/bin:/sbin:/bin"
export PATH="/usr/sbin:/usr/bin:/sbin:/bin"

# SHELL variable is used inside pbuilder by commands like 'su';
# and they need sane values
export SHELL="/usr/bin/bash"

# The name of debootstrap command, you might want "cdebootstrap".
DEBOOTSTRAP="debootstrap"
```

```
# default file extension for pkgname-logfile
PKGNAME_LOGFILE_EXTENSION="_$(dpkg --print-architecture).build"

# default PKGNAME_LOGFILE
PKGNAME_LOGFILE=""

# default AUTOCLEANAPTCACHE
AUTOCLEANAPTCACHE=""

#default COMPRESSPROG
COMPRESSPROG="gzip"

# pbuilder copies some configuration files (like /etc/hosts or
# /etc/hostname)
# from the host system into the chroot. If the directory specified here
# exists and contains one of the copied files (without the leading /etc) that
# file will be copied from here instead of the system one
CONFDIR="/etc/pbuilder/conf_files"
```

Diese Werte können gegebenenfalls von Werten in der Datei */etc/pbuilderrc* überschrieben werden.

Nach einer Neuinstallation sieht die */etc/pbuilderrc* wie folgt aus:

```
# this is your configuration file for pbuilder.
# the file in /usr/share/pbuilder/pbuilderrc is the default template.
# /etc/pbuilderrc is the one meant for overwriting defaults in
# the default template
#
# read pbuilderrc.5 document for notes on specific options.
MIRRORSITE=http://ftp.de.debian.org/debian/
```

Diese habe ich wie folgt eingerichtet:

```
# this is your configuration file for pbuilder.
# the file in /usr/share/pbuilder/pbuilderrc is the default template.
# /etc/pbuilderrc is the one meant for overwriting defaults in
# the default template
#
# read pbuilderrc.5 document for notes on specific options.
# adapt from Mechtilde - 2019-09-07 (analog wiki)
MIRRORSITE=http://ftp.de.debian.org/debian/
AUTO_DEBSIGN=${AUTO_DEBSIGN:-no}
HOOKDIR=/var/cache/pbuilder/hooks
# Codenames for Debian suites according to their alias.
# Update these when needed.
# EXPERIMENTAL_CODENAME = "experimental"
UNSTABLE_CODENAME="sid"
TESTING_CODENAME="bookworm"
STABLE_CODENAME="bullseye"
STABLE_BACKPORTS_SUITE="${STABLE_CODENAME}-backports"
```

Neben der globalen Konfigurationsdatei */etc/puilderrc* kann auch eine nutzerspezifische Datei *~/pbuilderrc* angelegt werden. Der Inhalt dieser Datei überschreibt die systemweiten Einstellungen.

```
# BINDMOUNTS is a space separated list of things to mount
# inside the chroot.
BINDMOUNTS="/var/local/repository" # lokales Verzeichnis einbinden (mounten)
OTHERMIRROR="deb http://deb.debian.org/debian/ buster-backports main \
| deb [trusted=yes] file:///var/local/repository ./"
# OTHERMIRROR="${OTHERMIRROR} | deb file:///var/local/repository ./"
# Fertige Pakete im lokalen Repository ablegen
# BUILDRESULT=..

# Added after reading:
# https://lists.debian.org/debian-backports/2018/09/msg00021.html

# List of Debian suites.
DEBIAN_SUITES=($UNSTABLE_CODENAME $TESTING_CODENAME \
$STABLE_CODENAME $STABLE_BACKPORTS_SUITE
"experimental" "unstable" "testing" "stable")
```

4. Mai 2025

```
# Mirrors to use. Update these to your preferred mirror.  
DEBIAN_MIRROR="ftp.de.debian.org"  
  
# Added after reading https://wiki.debian.org/cowbuilder  
BASEPATH="/var/cache/pbuilder/base.cow/"
```

Neben den systemweiten und nutzerspezifischen Konfigurationen werden auch paketspezifische Konfigurationen des Pbuilders benötigt. Diese Dateien können im Projektverzeichnis abgelegt und mit `--configfile <Konfigurationsdatei>` eingelesen werden. Mit dieser Konfiguration werden eventuell schon vorhandene Werte überschrieben. Hier können dann die Informationen zu den Projekten abgelegt werden, wenn für diese Veröffentlichungen im Backports-Zweig benötigt werden. Diese Datei wird nach allen anderen Konfigurationsdateien eingelesen.

Siehe dazu auf Seite

<https://wiki.debian.org/BuildingFormalBackports>

den Abschnitt *#Advanced: _Building_multi-dependency_packages*

Dies kann auch in der Datei `~/.pbuilderrc` hinzugefügt werden, wenn es diese Datei gibt. Als MIRROR wird ein gängiger gut erreichbarer Debian-Mirror angegeben. Falls vorhanden, kann hier auch die Adresse eines vorhandenen lokalen Spiegel eingetragen werden.

Für die *Hook*-Skripte ist ein Verzeichnis `~/.pbuilder/` anzulegen.

19.3.3. Hooks einrichten

Hooks sind Skripte, die an bestimmten, vordefinierten Punkten während des Bauprozesses Dinge erledigen. Mit den sogenannten *Hooks* (Haken) kann der Prozess in der Build-Chroot auch an vordefinierten Positionen unterbrochen werden, um noch manuell in den Prozess eingreifen zu können.

Die *Hook*-Skripte befinden sich im Verzeichnis `~/.pbuilder/`

Der Name des Hook-Skriptes bestimmt, an welcher Stelle im Bauprozess der *Hook* ausgeführt wird.

Dabei gilt folgende Konversation:

```
X<digit><digit><whatever-else-you-want-as-name>
```

Das „X“ (A bis G) bestimmt die Hook-Klasse, die folgenden 2 Ziffern die Reihenfolge, in der die Hooks einer Klasse ausgeführt werden. Der Rest dient als Beschreibung.

Leider entspricht die Reihenfolge, in der die Klassen im Build-Prozess ausgeführt werden, nicht der alphabetischen Reihenfolge.

- A** Ist für das `--build` Ziel. Es wird vor dem Baubeginn ausgeführt. D.h. nach dem Entpacken des Bausystems, des Quelltextes und nach der Erfüllung der Bau-Abhängigkeit.
- B** Wird ausgeführt, nachdem das Bausystem den Bau erfolgreich abgeschlossen hat, bevor das Bauergebnis zurückkopiert wird. - Unterbrechung nach erfolgreichem Bauen
- C** Wird nach einem Build-Fehler, vor der Bereinigung ausgeführt. - Unterbrechung nach gescheitertem Build
- D** Wird vor dem Entpacken der Quelle innerhalb der Chroot-Umgebung ausgeführt, nachdem die Chroot-Umgebung eingerichtet wurde. Erstellt `$TMP` und `$TMPDIR` wenn erforderlich. Dies wird aufgerufen, bevor die Build-Abhängigkeit befriedigt ist. Auch nützlich für den Aufruf von `apt update`. – Möglichkeit zum Editieren der `Sources.list`.²

²weiteres s. a. https://wiki.debian.org/PbuilderTricks#How_to_include_local_packages_in_the_build

- E** Wird ausgeführt, nachdem *pbuilder --update* und *pbuilder --create* die Arbeit von *apt-get* mit dem Chroot beendet hat, bevor das Kernel-Dateisystem (*/proc*) umountet und der Tarball aus dem Chroot erzeugt wird.
- F** Is executed just before user logs in, or program starts executing, after chroot is created in *--login* or *--execute* target.
- G** Is executed just after *debootstrap* finishes, and configuration is loaded, and *pbuilder* starts mounting */proc* and invoking *apt install* in *--create* target.
- H** Wird unmittelbar nach dem Auspacken von *chroot*, mounting *proc* und jedem in *BINDMOUNTS* angegebenen *bind mount* ausgeführt. Es wird für jedes Ziel ausgeführt, das die entpackte Chroot benötigt. Es ist nützlich, wenn Sie die Chroot-Einbauten dynamisch ändern wollen, bevor irgendetwas anfängt, sie zu benutzen.
- I** Wird ausgeführt, nachdem das Bausystem den Bau erfolgreich abgeschlossen hat, nach dem Zurückkopieren der Build-Ergebnisse.

19.3.4. Hooks - Beispiele

Diese stehen alle im Verzeichnis: *~/pbuilder/*

19.3.4.1. Hook A

Dieser Hook könnte z.B. *A10shell* heißen.

```
65a <Hook-A 65a>≡
#!/usr/bin/bash
# example file to be used with --hookdir
#
# invoke shell before build starts.

BUILDDIR="${BUILDDIR:-/tmp/buildd}"

apt-get install -y "${APTGETOPT[@]}" nano less
cd "$BUILDDIR"/*/debian/..
echo "Hook A - the dependencies are installed. Now the build can start."
echo "Please use CTRL-D to continue"
/usr/bin/bash < /dev/tty > /dev/tty 2> /dev/tty
```

19.3.4.2. Hook B

Dieser Hook könnte z.B. *B20shell* heißen.

```
65b <Hook-B 65b>≡
#!/usr/bin/bash
# example file to be used with --hookdir
#
# invoke shell if build fails.

BUILDDIR="${BUILDDIR:-/tmp/buildd}"

# apt-get install -y "${APTGETOPT[@]}" vim less
cd "$BUILDDIR"/*/debian/..
echo "Hook B - The build was built successfully"
echo "You can check it with ls -la ../"
/usr/bin/bash < /dev/tty > /dev/tty 2> /dev/tty
```

19.3.4.3. Hook C

Hier können dann noch Pakete hinzu installiert werden, die zwingend benötigt werden, wenn der Build fehlschlägt.

Dieser Hook könnte z.B. *C10shell* heißen.

```
66a <Hook-C 66a>≡
#!/usr/bin/bash
# example file to be used with --hookdir
#
# invoke shell if build fails.

BUILDDIR="${BUILDDIR:-/tmp/buildd}"

apt-get install -y "${APTGETOPT[@]}" vim less mc unzip locate
cd "$BUILDDIR"/*/debian/..
echo "Hook C - The build wasn't built successfully"
echo "After analysing the errors you can continue with using CTRL-D"
/usr/bin/bash < /dev/tty > /dev/tty 2> /dev/tty
```

19.3.4.4. Hook D

```
66b <Hook-D 66b>≡
#!/usr/bin/bash
# example file to be used with --hookdir
#
# invoke shell before unpacking the source
# inside the chroot

BUILDDIR="${BUILDDIR:-/tmp/buildd}"

apt-get install -y "${APTGETOPT[@]}" less nano
#cd "$BUILDDIR"/*/debian/..
echo "Hook D -"
echo "After unpacking the sources the dependencies"
echo "can be downloaded and unpacked."
echo "Please use CTRL-D to continue"
/usr/bin/bash < /dev/tty > /dev/tty 2> /dev/tty
```

19.3.4.5. Hook E

```
66c <Hook-E 66c>≡
echo "Hook E"
echo "Please use CTRL-D to continue"
/usr/bin/bash < /dev/tty > /dev/tty 2> /dev/tty
```

19.3.4.6. Hook F

```
66d <Hook-F 66d>≡
echo "Hook F"
echo "Please use CTRL-D to continue"
/usr/bin/bash < /dev/tty > /dev/tty 2> /dev/tty
```

19.3.4.7. Hook G

```
67a  <Hook-G 67a>≡
      echo "Hook G"
      echo "Please use CTRL-D to continue"
      /usr/bin/bash < /dev/tty > /dev/tty 2> /dev/tty
```

19.3.4.8. Hook H

```
67b  <Hook-H 67b>≡
      #!/usr/bin/bash
      # example file to be used with --hookdir
      #
      # invoke shell if build fails.

      BUILDDIR="${BUILDDIR:-/tmp/buildd}"

      echo "Hook H"
      echo "Executed after preparing the chroot \n and before installing the dependencies"
      echo "Here you can include dependency from e.g a local repo for testing."
      echo "Next the source code of the package to be built is unpacked."
      echo "Please use CTRL-D to continue"
      /usr/bin/bash < /dev/tty > /dev/tty 2> /dev/tty
```

19.3.4.9. Hook I

```
67c  <Hook-I 67c>≡
      #!/usr/bin/bash
      # example file to be used with --hookdir
      #
      # invoke shell if build fails.

      BUILDDIR="${BUILDDIR:-/tmp/buildd}"

      #apt-get install -y "${APTGETOPT[@]}" vim less
      #cd "$BUILDDIR"/*/debian/..
      echo "Hook I"
      echo "Please use CTRL-D to continue"
      /usr/bin/bash < /dev/tty > /dev/tty 2> /dev/tty
```

19.3.5. Alternative *Chroot*-Umgebungen

Es hat sich bewährt, für die verschiedenen *Debian*-Zweige eigene *Chroot*-Umgebungen bereitzustellen.

Dazu wird eine Kopie von dem Verzeichnis `/var/cache/pbuilder/base.cow` angelegt. Darin kann dann z.B. Die Datei `/etc/apt/sources.lists` entsprechend angepasst werden.

Dabei kann es bei einer Aktualisierung der Pakete notwendig sein, dass zunächst auch Abhängigkeiten dieser Pakete aktualisiert werden müssen. Diese stehen jedoch nur mit Zeitverzögerung im Repo zur Verfügung. Hier kann es helfen für die weiteren Tests schon einmal die Zeile

```
deb http://incoming.debian.org/debian-buildd buildd-unstable main
```

in der `/etc/apt/sources.lists` zu aktivieren. Damit stehen die dortigen Pakete für das Bauen in der *Pbuilder*-*Chroot* zur Verfügung.

19.4. Konfiguration von *sbuild*

sbuild kann ohne Root-Rechte genutzt werden. In diesem Fall ist der Paketbetreuer in die Gruppe *sbuild* aufzunehmen.

Für diese Nutzung sind folgende Pakete zu installieren.[34]

```
sudo apt install sbuild mmdebstrap uidmap
```

Die benötigten Verzeichnisse sind wie folgt zu installieren:

```
mkdir -p ~/.cache/sbuild
```

Nun wird der Tarball der Chroot erstellt.

```
mmdebstrap --variant=build unstable ~/.cache/sbuild/unstable-amd64.tar.zst
```

Danach wird eine neue Konfigurationsdatei *~/.sbuildrc* erstellt. Eine bereits bestehende Datei wird überschrieben. Sollten Sie bereits eine *~/.sbuildrc* haben, erstellen Sie bei Bedarf eine Sicherungskopie.

```
68 <.sbuild.rc 68>≡
    $chroot_mode = 'unshare';
    #distribution = 'unstable';
    $build_source = 1;
    $source_only_changes = 1;
    $verbose = 1;
    #run_autopkgtest = 1;
    $autopkgtest_root_args = '';
    $autopkgtest_opts = [ '--apt-upgrade', '--', 'unshare', '--release', '%r', '--arch', '%a' ];
```


19.5. Weitere Chroot-Systeme

Neben dem Bauen in einer *pbuilder*-Chroot gibt es noch weitere Situationen, in denen die Nutzung eines separaten Systems nützlich sein kann. Dies gilt besonders für das Ausführen von *mh_make*. (Kapitel 47.3, Seite 386)

Diese Einrichtung einer *Maven-Chroot* wird als Beispiel beschrieben:

Zunächst ist das Paket *debootstrap*, sofern noch nicht vorhanden, zu installieren. Dann wird mit

```
sudo mkdir --parents /srv/maven-chroot
```

ein entsprechendes Verzeichnis für die *Maven-Chroot* angelegt. Die Chroot selber wird mit

```
sudo /usr/sbin/debootstrap --arch amd64 sid \
/srv/maven-chroot http://ftp.de.debian.org/debian
```

erstellt.^[35]

Danach kann der Nutzer *root* mit dem Befehl *chroot* ein neues Wurzelverzeichnis starten.

Die konkreten Befehle lauten (als *root*):

```
# mount --options bind /proc /srv/maven-chroot/proc
# mount devpts /dev/pts --types devpts
# LANG=C chroot /srv/maven-chroot /usr/bin/bash
```

Die letzte Zeile startet die angelegte Chroot.

Dieser User *root* kann dann nicht mehr auf Dateien außerhalb des neuen Wurzelzeichnisses zugreifen.

Die Chroot-Umgebung kann wie folgt wieder beseitigt werden:

```
sudo umount /srv/maven-chroot/proc # Unmount first!
sudo rm -rf /srv/maven-chroot/
```

Der Ablauf des Programmskript wird auf Wunsch zur Nutzung einer *Maven-Chroot* angehalten (Kapitel 47.3, Seite 386).

19.6. Quilt fürs Patchen einrichten

Mit diesem Skript wird für das Patchen unter anderem *dquilt* verwendet. Dies ist eine *debian*-spezifische Anpassung für *quilt*.

Um diese Anpassung zu erzeugen, bedarf es der Datei *.quilttrc-dpkg* mit folgendem Inhalt ³:

```
69 <DQuilt 69>≡
d=. ; while [ ! -d $d/debian -a 'readlink -ev $d' != / ]; do d=$d/..; done
if [ -d $d/debian ] && [ -z $QUILT_PATCHES ]; then
# falls in Debian-Paketbaum mit ungesetztem $QUILT_PATCHES
QUILT_PATCHES="debian/patches"
QUILT_PATCH_OPTS="--reject-format=unified"
QUILT_DIFF_ARGS="-p ab --no-timestamps --no-index --color=auto"
QUILT_REFRESH_ARGS="-p ab --no-timestamps --no-index"
QUILT_COLORS="diff_hdr=1;32:diff_add=1;34:diff_rem=1;\
31:diff_hunk=1;33:diff_ctx=35:diff_cctx=33"
if [ ! [ -d $d/debian/patches ] ]; then mkdir $d/debian/patches; fi
fi
```

³<https://www.debian.org/doc/manuals/maint-guide/modify.html>

4. Mai 2025

Dazu gehört für den manuellen Betrieb auch der Eintrag in der Datei `~/.bashrc`. Dieser Eintrag sieht wie folgt aus:

```
# wird fuer die Patch-Verfolgung mit Quilt benoetigt
alias dqilt="quilt --quiltrc=${HOME}/.quiltrc-dpkg"
complete -F -quilt-completion $_quilt_complete_opt dqilt
```

Zu beachten ist, dass Einstellungen in der Datei `~/.bashrc` bei der Ausführung dieses Programmskriptes nicht beachtet werden. Es müssen also alle notwendigen Einstellungen im Programmskript komplett abgebildet werden.

Die Verwendung von *quilt* bzw. *dqilt* wird in Kapitel *Nutzung von Quilt* (Kapitel 36.5, Seite 276) beschrieben.

20. Git einrichten

20.1. Branches

Das Git-Repository eines Debian-Paketes hat in der Regel mindestens folgende Zweige:

- *debian/sid*
- *upstream*
- *pristine-tar*

Der Branch *debian/sid* kann auch *master* oder *main* genannt werden.

Es kann einen zusätzlichen Branch für *experimental* geben. Außerdem können Branches für *backports* und *update-proposal* angelegt werden.

Solche weiteren Branches können auch vom Programmskript angelegt werden. (Kapitel 44.1, Seite 375)

20.2. Mergen

Wird von *debian/experimental* nach *debian/sid* gemergt, wird in der Regel ein Fast-Forward-Merge durchgeführt.

Dies ist dann der Fall, wenn - wie hier - zunächst in *debian/experimental* weitere Aktualisierungen zum Testen eingepflegt wurden, dann aber die Entwicklung in *debian/sid* fortgeführt werden soll.

Wurden jedoch in der Zwischenzeit auch im Branch *debian/sid* Änderungen hinzugefügt, kann nur ein Recursive-Merge durchgeführt werden.

Gegebenenfalls müssen einige Anpassungen, z.B. im Verzeichnis *debian/* einzeln ausgewählt und dem jeweiligen Zweig hinzugefügt werden (*git cherry-pick*).

Eine dazu bewährte Befehlszeile ist:

```
git cherry-pick --edit -x <commit>
```

Sollen mehrere Commits auf einmal dem Zweig zugeführt werden, gilt folgende Befehlszeile:

```
git cherry-pick --no-commit <commit> <commit> \dots
```

20.3. *gbp.conf*

Das Programmskript nutzt die Anwendungen aus dem Debian-Paket *git-buildpackage*. *git-buildpackage* (*gbp*) kann und sollte konfiguriert werden.

Die Konfigurationsdatei *gbp.conf* wird zur Steuerung dieser Anwendung verwandt. Sie kann an verschiedenen Stellen im Dateisystem platziert werden.

20.3.1. Reihenfolge

Die Konfigurationsdateien für gbp werden in folgender Reihenfolge eingelesen:

1. /etc/git-buildpackage/gbp.conf, die systemweite Konfigurationsdatei
2. ~/.gbp.conf, die nutzerspezifische Konfigurationsdatei
3. debian/gbp.conf, Konfiguration für das Repositorium oder den Branch
4. git/gbp.conf, Konfiguration für das lokale Repositorium

Alle Konfigurationsdateien haben das gleiche Format. ¹

Durch Setzen der Umgebungsvariablen `GBP_CONF_FILES` kann diese Reihenfolge überschrieben werden. Der Inhalt dieser Variable kann mit `echo $GBP_CONF_FILES` ermittelt werden. ².

20.3.2. Abschnitte in der *gbp.conf*

Es gibt verschiedene Abschnitte in der *gbp.conf*. Diese Abschnitte sind alle optional.

Für jeden *gbp*-Befehl³ kann ein eigener Abschnitt erstellt werden. Zusätzlich gibt es einen Abschnitt, der für alle Befehle gilt

Einige wichtige Abschnitte werden im Folgenden aufgeführt.

[DEFAULT] In diesem Abschnitt angegebenen Optionen werden auf alle *gbp*-Befehle angewandt. ⁴

[import-orig] Die Optionen dieses Abschnittes überschreiben die entsprechenden Abschnitte unter *[DEFAULT]*. Sie werden auf den Befehl *gbp import-orig* angewandt.

[pq] Die Optionen dieses Abschnittes werden auf den Befehl *gbp pq* angewandt und überschreiben die des Abschnittes *[DEFAULT]*.

[dch] Die Optionen dieses Abschnittes werden auf den Befehl *gbp dch* angewandt und überschreiben die Optionen des Abschnittes *[DEFAULT]*.

[buildpackage] Die Optionen dieses Abschnittes werden auf den Befehl *gbp buildpackage* angewandt und überschreiben die Optionen des Abschnittes *[DEFAULT]*.

20.3.3. Syntax der Optionen

Die Optionen in den Abschnitten der *gbp.conf* werden aus den Befehlszeilenoptionen gebildet. Die möglichen Optionen zu den einzelnen *gbp*-Befehlen können der jeweiligen Manpage entnommen werden.

Diese werden ohne das einleitende doppelte Minuszeichen angegeben. Aus Befehlszeilenoption `--patch-num-format=%02d_` wird dann `patch-num-format=%02d_`.

Im Falle von *gbp buildpackage* ist zusätzlich auch *git-* wegzulassen⁵.

Der Eintrag `pbuilder-options=PBUILDER_OPTION` in der *gbp.conf* entspricht also `--git-pbuilder-options=PBUILDER_OPTION`.

20.3.4. Beispiel

Im Homeverzeichnis des Nutzers kann eine Datei `~/.gbp.conf` beispielsweise wie folgt angelegt werden:

72 `<gbp.conf 72>≡`

```
[DEFAULT]
sign-tags = True
```

¹[3], Abschnitt *Configuration Files* und die Manpage zu *gbp-conf*

²[3] Abschnitt *Configuration Files/Overriding Parsing Order*

³[3] Manpage zu *gbp-conf*

⁴[3] Manpage zu *gbp-conf*

⁵Manpage zu *gbp-conf*[3]

```
# keyid for signing the package
keyid = 0x<keyid>
pristine-tar = True
# generate gz compressed orig file
# compression = xz
# If you want to use normally sbuild
# builder = sbuild

[buildpackage]
postbuild = lintian $GBP_CHANGES_FILE
cleaner = /bin/true
# If you want to use normally pbuilder
# pbuilder = True
pbuilder-options = --source-only-changes --hookdir /home/mechtilde/.pbuilder

#[buildpackage]
# use a build area relative to the git repository
# export-dir=./build-area
# to use the same build area for all packages use an absolute path:
#export-dir=/home/debian-packages/build-area

[dch]
id-length = 7

# Options only affecting gbp pq
[pq]
#patch-numbers = False
# The format specifier for patch number prefixes
#patch-num-format = '%04d-'
patch-num-format = '%02d_'
# Whether to renumber patches when exporting patch queues
#renumber = False
renumber = True
# Whether to drop patch queue after export
#drop = False
```

4. Mai 2025

In der Datei `/etc/git-buildpackage/gbp.conf` werden die Konfigurationsmöglichkeiten aufgeführt.

20.4. Git-Repositoryn auf eigener Infrastruktur

20.4.1. Lokales Git-Repository

Das lokale Git-Repository wird entweder vom Skript angelegt (Kapitel 32.4, Seite 135) oder durch Klonen erzeugt (Kapitel 32.5, Seite 150). Außerdem kann es mit `gbp import-dsc` erzeugt werden (Kapitel 32.6, Seite 156).

Ihm können weitere Branches hinzugefügt werden (Kapitel 44.1, Seite 375).

20.4.2. Eigener Git-Server

Das lokale Git-Repository kann auch auf einem eigenen Git-Server „gespiegelt“ werden.

Die Einrichtung des Servers erfolgt manuell. Die Verwendung von `cgit` oder `gitweb` erleichtern den Zugriff.

Im Programmskript können der Name oder die IP des eigenen Git-Servers eingegeben werden (Kapitel 44.2, Seite 376).

Der „Workflow“ ist dann wie folgt: Nach der Anlage der Konfigurationsdatei wird zunächst der Name oder die IP des eigenen Git-Servers eingegeben (Kapitel 44.2, Seite 376), bevor mit dem Bauen eines neuen Paketes begonnen wird (Kapitel 32, Seite 111).

Das „fertige“ Paket kann dann auf den eigenen Git-Server hochgeladen werden (Kapitel 42.2, Seite 355).

21. Salsa-Repositorien

Salsa ist der Name eines gemeinschaftlichen Entwicklungsserver für *Debian*, der auf der *GitLab*-Software basiert. *Salsa* soll die notwendigen Werkzeuge für die kollaborative Entwicklung für Paketbetreuer, Paketierungsteams und andere *debian*-bezogene Einzelpersonen und Gruppen bereitstellen.[36]

Salsa bietet alle Funktionen von *GitLab*. Der Dienst steht unter <https://salsa.debian.org> zur Verfügung. Es gibt eine Dokumentation der *debian*-spezifischen Eigenheiten [37], mit denen man sich zunächst vertraut machen sollte.

21.1. Salsa-Account anlegen

Das Anlegen eines Accounts auf *Salsa* ist sehr einfach. Man ruft die Seite https://salsa.debian.org/users/sign_up auf, welche selbsterklärend ist.

21.2. Anlage eines Salsa-Repositoriums

Ein *Git*-Repositorium auf *salsa.debian.org* wird nicht vom Programmskript eingerichtet.

Nach dem Login und der Authentifizierung auf <https://salsa.debian.org> kann dort im jeweiligen Team ein neues Repositorium angelegt werden.

Für die Anlage von neuen Projekten auf <https://salsa.debian.org> sind entsprechende Rechte erforderlich. Dies können die Rechte eines *Debian*-Developers sein. Für die team-betreuten Projekte können diese Rechte auch an weitere Personen von den Betreuenden vergeben werden.

Eine Beschreibung der Anlage eines Projektes im Java-Team erfolgt in Kapitel 21.3, Seite 76.

Nach dem Login auf der Seite <https://salsa.debian.org> und der Auswahl des passenden Projektes für das neue Paket, wird mit dem Klick auf den Button *Neu erstellen* und der Auswahl der Funktion *Neues Projekt/Repository* die entsprechende Seite aufgerufen. Dort ist meist *Ein leeres Projekt* auszuwählen. Dann gibt man den Projektnamen ein. Dies ist meist der Name des Quellcodepaketes. Als Sichtbarkeitsgrad (Visibility Level) wird *Öffentlich* ausgewählt. Dann folgt der Klick auf den Button *Projekt anlegen*.

Auf der folgenden Web-Seite gibt es noch einige Erläuterungen. Vieles davon wird zunächst lokal mit dem beschriebenen Programm angelegt.

In der linken Navigationsleiste werden nun im Bereich *Einstellungen* weitere Konfigurationen zum Projekt vorgenommen.

Hier ist es wichtig unter *CI/CD* den Pfad und den verwendeten Dateinamen anzugeben.

CI steht für Continuous Integration

CD steht für Continuous Delivery

Die hier verwendete Datei heißt *salsa-ci.yml* (Kapitel 35.4.9, Seite 247) im Verzeichnis *debian/*.

Das Build-Skript trägt das *Salsa*-Repositorium als „Remote-Repositorium“ ein und erinnert den Nutzer an die Anlage auf *salsa.debian.org* (Kapitel 32.4.3, Seite 148)

21.3. Salsa-Repositorium für das Java-Team

Salsa-Repositorien, die einem speziellen Projekt (wie beispielsweise dem Java-Team) zugeordnet sind, sollten möglichst einheitlich angelegt werden. Häufig wird hierzu ein Skript vom Projekt bereitgestellt, welches hierzu verwandt werden soll.

Dazu wechselt man in das gewünschte Team-Verzeichnis.

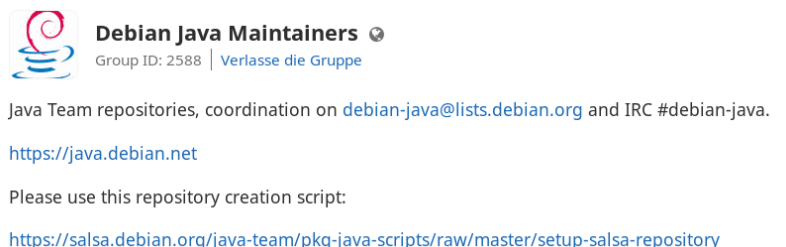


Abbildung 21.1.: Information zum Java-Team^a

^aQuelle: <https://salsa.debian.org/java-team/>

21.3.1. Quelle des Skripts

Unter

<https://salsa.debian.org/java-team/pkg-java-scripts/blob/master/setup-salsa-repository> kann das Skript des *Java-Teams* heruntergeladen werden. Es ist auch im Anhang beigefügt (Kapitel 50.1, Seite 409).

21.3.2. Abhängigkeiten

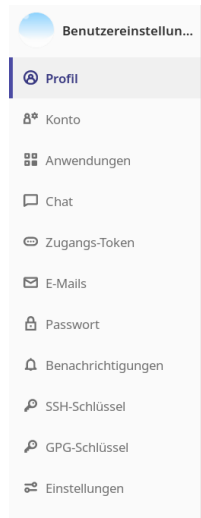
Dieses *Setup-Skript* nutzt *jq*. *jq* ist ein leichtgewichtiger und flexibler *JSON*-Prozessor für die Kommandozeile. Er hat nur minimale Laufzeitabhängigkeiten. Es gibt ein gleichnamiges Debian-Paket. Dieses Paket muss lokal installiert sein (Kapitel 19.1.2, Seite 58).

21.3.3. Zugangstoken beschaffen

Für die Anpassung wird noch ein projektspezifischer Token benötigt.

Nach dem Login auf salsa.debian.org wird im Dropdown-Menü des Nutzers *Einstellungen* gewählt. Damit wird die Seite *Benutzereinstellungen* des eingeloggten Nutzers aufgerufen.

In der linken Leiste befindet sich nun der Eintrag *Zugangs-Token*. Dort wird die Seite https://salsa.debian.org/profile/personal_access_tokens zur Erstellung eines solchen Token aufgerufen. Er wird für jedes Projekt neu generiert.

Abbildung 21.2.: Zugangstoken erstellen^a

^aQuelle:<https://salsa.debian.org>

Dort wird der Name des zu erstellenden Projektes eingegeben. Ferner wird ein Ablaufdatum für das Token eingegeben. Danach wird der Gültigkeitsbereich des Tokens festgelegt. Als *Scope* ist hier *api* auszuwählen. Mit dem Klick auf den Button *Create personal access token* erscheint oben auf der Seite der Zugangstoken.

21.3.4. Token eintragen

Das generierte Token ist als *SALSA_TOKEN* in das Skript einzutragen. Das Kommentarsymbol ist zu entfernen

Es erscheint sinnvoll, dieses Skript jeweils im Projektverzeichnis abzulegen.

21.3.5. Skript aufrufen

Das Skript wird nun mit dem Namen des neuen Projektes (Paketes) als Parameter aufgerufen:

```
./setup-salsa-repository.sh <packagename>
```

Danach werden folgende Meldungen ausgegeben (anhand des Beispiels BeanValidationApi):

```
./setup-salsa-repository.sh beanvalidation-api
Creating the beanvalidation-api repository\dots
Configuring the BTS tag pending hook\dots
Configuring the KGB hook\dots
Configuring email notification on push\dots
```

```
Done! The repository is located at
https://salsa.debian.org/java-team/beanvalidation-api
```

Erscheint lediglich die erste Zeile, sind das verwendete Skript (s. Kapitel 50.1, Seite 409) und Token zu prüfen.

4. Mai 2025

21.4. Aufgaben auf *salsa.debian.org*

21.4.1. Merge Request

Manchmal gibt es auch sogenannte Merge Request, in dem andere Patches bereitstellen.

Auf *salsa.debian.org* gibt es in der linken Navigationsleiste dann einen Eintrag „Merge-Requests“. Dort kann dieser dann bearbeitet werden.

Dies erfolgt durch Klicken auf die Commit-Message.

22. Paketieren jenseits vom Zweig *Unstable*

Es gibt verschiedene Gründe, warum von dem generellen Weg, neue Upstream-Versionen ausschließlich für *Unstable = sid* zu paketieren, abgewichen werden darf und wird. Auch das Programmskript ermöglicht dies (Kapitel 39, Seite 325).

Die Entwicklerreferenz [9] bezeichnet das Hochladen in die Distribution *Stable* und *Oldstable* als „Sonderfall“.¹

Ein wesentlicher Grund ist das Vorhandensein eines schwerwiegenden Fehlers oder eines Sicherheitsproblems. Schwerwiegende Fehler in einem Paket werden in der Regel durch einen entsprechend eingestuften Fehlerbericht gemeldet.

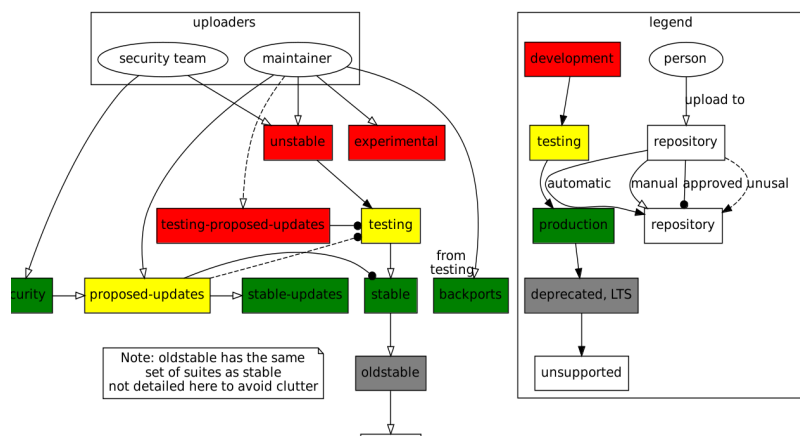
Eine weitere Abweichung von der generellen Regel gibt es bezüglich der Pakete der *Mozilla-Suite*, *Firefox* und *Thunderbird*. Diese Pakete werden zeitnah nach ihrem Upstream-Release für die *ESR = Extended Support Release* Version auch als *Security-Updates* (Kapitel 22.1, Seite 80) für das *Stable*-Release bereitgestellt.

Daraus können sich z.B. für die Erweiterungen, hier als Webextensionen (Kapitel 14, Seite 47) beschrieben, Inkompatibilitäten mit der dann aktuellen Version im *Stable*-Release ergeben. Dies ist ebenfalls ein Grund, eine Aktualisierung im *Stable*-Release vorzunehmen (s.a. Kapitel 22.2.3, Seite 82).

Daneben ist es bei Desktop-Anwendungen oft wünschenswert, aktuellere Versionen derselben in einer stabilen Betriebssystemumgebung zu nutzen. Dazu werden die gewünschten Pakete als sogenannte *Backports* (d.h. Rückportierungen) zur Verfügung gestellt. (s. Kapitel 22.3, Seite 82)

Es gibt auch gute Gründe, Pakete zunächst nach *Experimental* (auch *rc-buggy* genannt) hochzuladen. Dies gilt besonders für neue Pakete. In dem Zeitraum, in dem die bestehenden Versionen „eingefroren“ sind und nur Fehlerbehebungen für das nächste Release erlaubt sind, werden neue Versionen oft vorab nach *Experimental* hochgeladen. (s. Kapitel 22.5, Seite 83)

¹Kapitel 5.5.1 in der Developer-Reference[9]

Abbildung 22.1.: Arbeitsabläufe [38] ^a

^a©2016 Antoine Beaupré anarc@debian.org, CC-BY-SA 4.0

22.1. Security-Updates

Wichtig ist die Rückportierung von Änderungen an Quellcode-Dateien wegen Sicherheitsproblemen.

Immer, wenn ein Sicherheitsproblem bekannt wird, soll der Maintainer mit dem Sicherheits-Team zusammenarbeiten, um für das *Stable*-bzw. *Oldstable*-Release eine korrigierte Version bereitzustellen. Die Fehlerbehebung sollte gezielt erfolgen. Patches sollten so klein wie möglich gehalten werden. Weitere Informationen befinden sich in den Kapiteln 3.1.2 und 5.8.5 der *Debian-Entwicklerreferenz*[9].

22.2. (Old-)Stable-Proposal

Sollen Pakete mit schwerwiegenden Fehlern, die nicht die IT-Sicherheit betreffen, aktualisiert werden, kann dies unter bestimmten Voraussetzungen in *Proposed-Updates* erfolgen. Gleiches gilt hinsichtlich der Erweiterungen für **Firefox** und **Thunderbird**.

Als Kriterien für das Hinzufügen von Paketen zu *stable-updates* sind folgende genannt worden. ²:

- Die Aktualisierung ist dringend und nicht sicherheitsrelevant. Die Sicherheitsaktualisierungen erfolgen im oben genannten Wege (Kapitel 22.1, Seite 80).
- Das fragliche Paket ist ein Datenpaket, und die Daten müssen zeitnah aktualisiert werden (z.B. tzdata).
- Korrekturen an Paketen, die durch externe Änderungen beeinträchtigt sind und von denen kein anderes oder nur wenige andere Pakete abhängig sind.
- Pakete, die aktuell sein müssen, um nützlich zu sein (z.B. clamav).

Wer glaubt, dass die angestrebte Aktualisierung diese Kriterien erfüllt, sollte sich vertrauensvoll an das Release Team über die Mailingliste debian-release@lists.debian.org, und erläutern, dass diese Aktualisierung auch über *stable-updates* erfolgen sollte.

Das für die Rückportierung auf diesem Wege vorgesehene Paket sollte bereits in *Unstable* oder besser noch in *Testing* veröffentlicht worden sein.

Während Backports in einem eigenen Repository gesammelt werden, werden *Proposed-Updates* in das *Stable-Proposed-Repository* eingefügt. Diese Pakete werden dazu

²<https://lists.debian.org/debian-devel-announce/2011/03/msg00010.html>

von den Debian-Entwicklern nach *Stable* hochgeladen. Dies gilt für Oldstable-Proposed-Updates entsprechend. Die Veröffentlichung erfolgt im nächsten Point-Release.

22.2.1. Fehlerbericht

Eine notwendige Voraussetzung ist auch hier zunächst ein aussagekräftiger Fehlerbericht (Bug-Report) (s.a. Kapitel 23, Seite 85). Dieser wird an `submit@bugs.debian.org` gesandt. Vorzugsweise sollte dafür das Programm `Reportbug` genutzt werden.

Dieser Fehlerbericht benötigt folgende Parameter:

```
Package: release.debian.org
Severity: normal
Tags: <ReleaseName>
User: release.debian.org@packages.debian.org
Usertags: pu
X-Debbugs-Cc: <SourceName>@packages.debian.org, <Maintainer>@debian.org
Control: affects -1 + src:<SourceName>
```

[Reason]

This package is ... # Why should it be updated?

[Impact]

Otherwise ... # What happens instead?

[Risks]

Which packages are affected?

[Checklist]

- [X] **all** changes are documented in the `d/changelog`
- [X] I reviewed all changes and I approve them
- [X] attach `debdiff` against the package in `(old)stable`
- [X] the issue is verified as fixed in `unstable`

[Changes]

What are the changes?

[Other info]

Do you have further infos?

Dieser Fehlerbericht wird gegen das Pseudopaket `release.debian.org` erstellt. Der Schweregrad eines solchen Fehlerberichtes ist im Allgemeinen maximal *normal*. Im Betreff wird der Name des Paketes aufgeführt, das für *Proposed-Updates* gebaut wird.

Proposed-Updates werden, wie dargelegt, nur unter besonderen Umständen zugelassen. Das *Release-Team* entscheidet, ob die Veröffentlichung erfolgt.

Daher muss der Fehlerbericht eine Begründung enthalten, warum diese Version in *stable* aktualisiert werden soll. Sofern es bereits eine dazugehörige Fehlernummer gibt, ist diese anzugeben. Dieser Fehlerbericht **muss** einen Schweregrad von *important* oder höher aufweisen.

Ein entsprechender Fehlerbericht muss gegebenenfalls auch für *Old-Stable* erstellt werden.

Weist dieser Fehlerbericht den notwendigen Schweregrad nicht auf, ist er zu berichtigen. Wie dies erfolgt, wird in Kapitel 24 (Seite 89)

22.2.2. Anforderungen an einen Patch

Ein Fehler sollte mit einem möglichst kleinen und passgenauen Patch behoben werden können. Damit soll verhindert werden, dass neue Fehler entstehen. Es dürfen auch keine neuen Abhängigkeiten hinzukommen. Auch muss berücksichtigt werden, welche anderen Pakete von diesem Paket abhängen.³

Diese passgenauen Patches werden mit `debdiff <BisherigesPaket>.dsc <NeuesPaket>.dsc > <Dateiname>.txt` dokumentiert. Es wird die Differenz angegeben, die zwischen der aktuellen Paketversion in Stable/Oldstable (bisheriges Paket) und der Version besteht, die hochgeladen werden soll (neues Paket). Wie diese Dokumentation mit dem Programmskript erstellt werden kann, wird in Kapitel 40.6.1 (Seite 335) beschrieben.

22.2.3. Abhängigkeiten zu Mozilla-Paketen

Eine hinreichende Begründung für *Webextensions* kann auch die oben (s. Kapitel 22, Seite 79) erwähnte Ausnahmeregelung für die Pakete der Mozilla-Suite sein.

Für diese Erweiterungen können sich nämlich Inkompatibilitäten mit der dann aktuellen Version von Firefox oder Thunderbird im *Stable*-Release ergeben. Dann sind die bisherigen Versionen unbrauchbar, was vor allem im produktiven Betrieb ein beachtliches Problem darstellt. (Kapitel 39, Seite 325)

22.3. Stable-Backports

Um Pakete für Backports bauen zu können, sind einige Vorbereitungen zu treffen.

Oft lässt sich ein solches Paket nicht ohne Weiteres in einer *Stable*-Umgebung bauen.

Dann werden weitere Pakete aus Backports benötigt. Pakete aus Backports werden aber nicht automatisch auch in einer solchen *Chroot* installiert.

Daher ist es notwendig, in `/etc/apt/preferences` eine solche Installation zu ermöglichen.

Dazu werden im *pbuilder Hooks* verwendet (Kapitel 19.3.3, Seite 64).

https://wiki.debianforum.de/Pbuilder_-_personal_package_builder

22.4. Backports-Repositorium

Regelmäßig werden die Pakete für den Unstable-Zweig gebaut. Von dort gelangen sie dann - sofern keine Fehler gefunden werden - in den Testing-Zweig. Beim Release wird der Testing-Zweig zunächst eingefroren und dann zum neuen Stable-Zweig. Es wird dann ein neuer Testing-Zweig eröffnet.

Werden Programmversionen aus *testing* oder (ausnahmsweise⁴) auch aus *unstable* in ein früheres Release übertragen, nennt man dies *backporting*.

Durch die an der Praxis orientierten Releasezyklen der stabilen Debian-Version ist es manchmal wünschenswert, Softwarepakete oder neuere Versionen aus Testing auch unter Stable verfügbar zu haben.

³Kapitel 5.5.1 in der Developer-Reference[9]

⁴in der Regel nur Sicherheits-Updates

Hierzu dient das Backports-Repositorium.
Versionierung (z.B. +deb9u1)
Backports gibt es für das Stable- und Oldstable-Release.

22.5. Experimental

22.6. Versionierung

Die Ursache bzw. Herkunft der Aktualisierung hat auch Einfluss auf die Versionierung. Dabei ist sicherzustellen, dass *dpkg* neuere Versionen korrekt als Upgrades interpretieren kann. Ein Blick auf die nächste zu erwartende Versionsnummer kann dabei helfen.

Beim Bauen für *experimental* wird in der Datei *debian/changelog* folgender Versionseintrag empfohlen:

<Versionsnummer>-<Revisionsnummer>~exp<Laufende Nummer>

Die Revisionsnummer ist in der Regel: 1

Beim Bauen für *Proposed-Updates* sind zwei Fälle zu unterscheiden.

Fall A Eine im *Stable*-Release vorhandene Version soll unter Beibehaltung dieser Versionsnummer korrigiert werden.

Fall B Ausnahmsweise soll eine neue Version in das Release aufgenommen werden.

Für den Versionseintrag in *debian/changelog* ist zwingend folgende Nomenklatur zu verwenden:

Fall A <Ursprüngliche Versions- und Revisionsnummer> +deb<Debian-Release-Nummer>u<Revisionsnummer des Updates>

Fall B <Versions- und Revisionsnummer aus Unstable/Testing>~deb<Debian-Release-Nummer>u<Revisionsnummer des Updates>

Die Verwendung der ~bewirkt, dass die Version in *Stable* kleiner als die in *Testing* (für das nächste Release) ist. Dies sichert den Aktualisierungspfad.

23. Zum Start eine E-Mail

Bevor das Paketieren mit dem Ziel der Veröffentlichung im Debian-Repositorium beginnen kann, bedarf es einer E-Mail. Diese löst einen „Bugreport“ aus, dessen Nummer in der Datei *debian/changelog* zu vermerken ist. Durch die Veröffentlichung des Paketes soll nämlich dann dieser Bugreport geschlossen werden.

Weitere Informationen dazu gibt es unter <https://www.debian.org/devel/wnpp/>

23.1. ITP - Intent To Package

ITP bedeutet „Intent to Package“. Dies bedeutet, dass an diesem Paket gearbeitet wird.

Dies ist die Bezeichnung eines Bug-Reportes, der gegen das Paket *WNPP* erstellt wird, was *Work-Needing and Prospective Package* bedeutet. Dies kann mit *Arbeitsbedürftige und voraussichtliche Pakete* übersetzt werden.

Das Paketieren von neuer Software sollte rechtzeitig angekündigt werden. Vielleicht gibt es ja noch Hinweise, was bei dem geplanten Paket zu beachten ist. Auch wird so verhindert, dass verschiedene Gruppen versuchen, dieselbe Software für Debian zu paketieren.

Eine Möglichkeit ist, dies mit dem auf jedem Debian-System installierten Tool *reportbug* durchzuführen. Es gibt aber auch die Möglichkeit einfach eine E-Mail-Vorlage zu nutzen. Die folgende Vorlage basiert auf dem *itp_template* mit dem *reportbug* diese E-Mail erstellt.

```
To: submit@bugs.debian.org
Subject: ITP: <Source Name> - <Short Description>
Package: wnpp
Severity: wishlist
```

```
* Package name : <Package Name>
  Version : x.y.z
  Upstream Author : Name <somebody@example.org>
* URL : http://www.example.org/
* License : (GPL, LGPL, BSD, MIT/X, etc.)
  Programming Lang: (C, C++, C#, Perl, Python, etc.)
  Description : <Short Description>
```

(Include the long description here.)

<And answer following questions:>

- * Why is this package useful/relevant?
- * Is it a dependency for another package?
- * Do you use it yourself?
- * If there are other packages providing similar functionality, how does it compare?
- * How do you plan to maintain it? Do you plan to maintain it

4. Mai 2025

```
inside a packaging team?  
(check list at https://wiki.debian.org/Teams)  
* Are you looking for co-maintainers? Do you need a sponsor?
```

Der Text in der ersten Zeile hinter dem *To:* kommt in die Adresszeile. Der Text in der zweiten Zeile hinter dem *Subject:* kommt in die Betreffzeile, wobei die Platzhalter durch den Namen des Quellcodes und einer kurzen Beschreibung ersetzt werden.

Wird das Paket hochgeladen, muss diese Nummer aus dem Bug-Tracking-System im *debian/changelog* vermerkt werden (Closes:#XXXXXX)

23.2. RFP - Request For Package

RFP bedeutet *Request for Package*. Dies bedeutet, dass ein solches Paket in Debian erwünscht ist.

Auch hierzu habe ich einmal eine E-Mail-Vorlage erstellt.

```
To: submit@bugs.debian.org  
Subject: RFP: <Sourcecode Name> - <Short Description>  
Package: wnpp  
Severity: wishlist
```

```
* Package name : <Package Name>  
  Version      : x.y.z  
  Upstream Author : Name <somebody@example.org>  
* URL          : http://www.example.org/  
* License      : (GPL, LGPL, BSD, MIT/X, etc.)  
  Programming Lang: (C, C++, C#, Perl, Python, etc.)  
  Description    : <Short Description>
```

(Include the long description here.)

<And answer following questions:>

```
* Why is this package useful/relevant?  
  Is it a dependency for another package?  
* Do you use it yourself?  
* If there are other packages providing similar functionality,  
  how does it compare?  
* How do you plan to maintain it? Do you plan to maintain it  
  inside a packaging team?  
  (check list at https://wiki.debian.org/Teams)  
* Are you looking for co-maintainers? Do you need a sponsor?
```

Auch hier gilt: Der Text in der ersten Zeile hinter dem *To:* kommt in die Adresszeile. Der Text in der zweiten Zeile hinter dem *Subject:* kommt in die Betreffzeile, wobei die Platzhalter durch den Namen des Quellcodes und einer kurzen Beschreibung ersetzt werden.

23.3. ITA - Intent To Adoption

Dies wird verwendet, wenn ein Paket, dass mit „O“ oder „RFA“ gekennzeichnet ist, übernommen werden soll.

Dazu muss der vorherige Fehlerbericht umbenannt und „O“ bzw. „RFA“ durch „ITA“ ersetzt werden.

Dabei tragen Sie sich als Besitzer ein.

Soll ein Fehlerbericht umbenannt oder der Besitzer geändert werden, muss dies per E-Mail an control@bugs.debian.org oder direkt an den Fehlerbericht über die Nummer (xxxxxx@bugs.debian.org) erfolgen¹.

Dabei ist ein strukturierter *Pseudo-Header* zu nutzen.²

23.4. RFA - Request for Adoption

23.5. RFH - Request For Help

23.6. O - Orphaned

23.7. RFS - Request For Sponsor

Wie auf <https://mentors.debian.net/sponsor/rfs-howto> beschrieben, wurde die Vorlage angepasst.

```
To: submit@bugs.debian.org
Subject: RFS: <Package Name> - <Short Description>
Package: sponsorship-request
Severity: normal
        [important for RC bugs, wishlist for new packages]
```

Dear mentors,

I am looking for a sponsor for my package "<Source Name>":

```
* Package name : <Source Name>
  Version : x.y.z
  Upstream Author : Name <somebody@example.org>
* URL : http://www.example.org/
* License : (GPL, LGPL, BSD, MIT/X, etc.)
  Programming Lang: (C, C++, C#, Perl, Python, etc.)
  Description : <Short Description>
```

It builds those binary packages:

<Name of the Binaries>

To access further information about this package, please visit the following URL:

<https://mentors.debian.net/package/<package name>>

Alternatively, one can download the package with `dget` using this command:

¹<https://www.debian.org/devel/wnpp/>

²<https://www.debian.org/Bugs/Reporting#control>

4. Mai 2025

```
dget -x https://mentors.debian.net/debian/pool/main/<p> \  
/<package name>/<package name>_x.y.z.dsc
```

Changes since the last upload:
[your most recent changelog entry]

Regards,

24. Änderungen am Fehlerbericht

Im Laufe eines solchen Prozesses kann es vorkommen, dass Änderungen am Fehlerbericht erfolgen müssen. Eine solche Änderung kann eine Änderung des Maintainers, des Titels oder auch Anderes sein.

Dazu werden die Befehle des *Kontroll-E-Mail-Server* genutzt. Unter <https://www.debian.org/Bugs/server-control> findet sich die Beschreibung dazu.

Dies kann mit einer strukturierten E-Mail erfolgen. Diese wird an die Adresse *control@bugs.debian.org* gesandt.

Befehle, die eigentlich an *control@bugs.debian.org* zu senden sind, funktionieren auch, wenn sie an *submit@bugs.debian.org* oder an *<bug number>@bugs.debian.org* gesandt werden ¹

24.1. Anpassung des Schweregrades

Im Betreff kann hier *Change severity* angegeben werden.

```
severity <bug number> important  
thank you
```

Für die automatische Auswertung der relevanten Informationen in der E-Mail muss sie mit *thanks*, *thankyou*, *thank you* oder einer anderen Endemarkierung wie *quit* oder *stop* abgeschlossen werden.

24.2. Anpassung des Titels

Für die Änderung des Titels wird dann zusätzlich eine Zeile wie folgt

```
retitle <bug number> -1 <neuer Titel>
```

eingefügt.

24.3. Änderung des Maintainers

Für die Änderung des Maintainers wird die folgende Zeile hinzugefügt:

```
Control: owner -1 <Neuer Maintainer oder wnpp@debian.org>
```

24.4. Öffnen eines geschlossenen Fehlerberichtes

Gegebenenfalls muss ein fälschlich geschlossener Bericht wieder geöffnet werden. Dies erfolgt mit

```
reopen Fehlernummer [ <Urheber-Adresse> | = | ! ]
```

¹<https://www.debian.org/Bugs/Reporting#control>

4. Mai 2025

24.5. Schließen eines Fehlerberichtes

```
close Fehlernummer [ Urheber-Adresse | = | ! ]
```

24.6. *usertags* hinzufügen

Im Laufe eines Maintainer-Lebens kommt es immer wieder vor, dass Fehlermeldungen mit tags versehen werden müssen oder sollten.

Z. B. ist es hilfreich für sogenannten *Bug Squashing Parties*, die geplanten und durchgeführten Fehlerbehebungen auch zu kennzeichnen.

Dafür wird eine E-Mail an den Fehlerbericht geschrieben. Adresse lautet dann `<Bugnummer>@debian.org`. Diese E-Mail soll auch *CC* an `control@bugs.debian.org` gehen.

Am Anfang einer solchen E-Mail steht dann:

```
user debian-release@lists.debian.org
usertags -1 + <Titel der BSP>
thank you
```

25. Reportbug einrichten

Das Comand-Line-Interface ist bereits Bestandteil der Basisinstallation. Zusätzlich kann mit dem Paket *reportbug-gtk* noch ein graphisches Nutzerinterface installiert werden.

26. Schwierigkeiten überwinden

26.1. Ein Paket loseisen

Eine Schwierigkeit ergibt sich daraus, dass es vor einem geplanten Release eine Zeitspanne gibt, in der die Pakete nicht mehr automatisch von *unstable* nach *testing* migrieren.

Im vollständigen „Freeze“ benötigen alle Pakete, die noch von *unstable* nach *testing* migrieren sollen, eine Entsperrung durch das Release-Team.[39]. Diese muss mit einem *Unblock Bugreport* beantragt werden.

26.1.1. Beantragung einer Entsperrung

Dazu wird zunächst eine Datei mit *debdiff* erstellt (s.a. Kapitel 22.2, Seite 80).

Damit wird die Differenz zwischen der Version in *testing* (alte Version) und *Unstable* (neue Version) mit der jeweiligen *dsc* erstellt.

Diese Differenz-Datei ist darauf zu prüfen, dass sie keine unwichtigen Änderungen für die gewünschte Fehlerbehebung hat.

Anschließend wird mit dem Werkzeug *reportbug* ein Fehlerbericht gegen das Paket *release.debian.org* erstellt und die Differenz-Datei angehängt. Dieser Fehlerbericht enthält eine ausführliche Begründung für die Änderungen und Verweise auf Fehlernummern. Ebenso enthält sie eine prägnante Beschreibung des Problems, das behoben wurde.

Dabei sollte auf folgende Fragen eingegangen werden.

```
Package: release.debian.org
User: release.debian.org@packages.debian.org
Usertags: unblock
Severity: normal
```

Please unblock package <source name>

(Please provide enough (but not too much) information to help the release team to judge the request efficiently. E.g. by filling in the sections below.)

[Reason]
(Explain what the reason for the unblock request is.)

[Impact]
(What is the impact for the user if the unblock isn't granted?)

[Tests]
(What automated or manual tests cover the affected code?)

[Risks]
(Discussion of the risks involved. E.g. code is trivial or complex, key package vs leaf package, alternatives available.)

4. Mai 2025

[Checklist]

- [] all changes are documented in the d/changelog
- [] I reviewed all changes and I approve them
- [] attach debdiff against the package in testing

[Other info]

(Anything else the release team should know.)

26.2. Releasekritische Fehler beheben

Vor einer neuen Veröffentlichung ist es oft notwendig, die Maintainer dabei zu unterstützen, Fehler zu beheben, die einer Veröffentlichung des Paketes entgegenstehen.

Dies geschieht häufig auf dafür organisierten Veranstaltungen ¹.

Unter

<https://www.debian.org/doc/manuals/developers-reference/pkgs.html#non-maintainer-uploads-nmus>
ist das gewünschte Vorgehen beschrieben.

Wesentlich ist dabei, dass in der Datei *debian/changelog* (Kapitel 37.1, Seite 291) ein entsprechender Eintrag in der zweiten Zeile erfolgt.

```
Non-maintainer upload
```

26.3. Paket aus Repositorien entfernen

In der Developer-Reference ² wird auch beschrieben, wie ein Paket entfernt werden kann.

Dazu muss zunächst festgestellt werden, dass kein weiteres Paket dieses als Abhängigkeit benötigt.

26.3.1. Feststellung der Rückwärtsabhängigkeit

Zu prüfen ist, ob andere Pakete das zu entfernende Paket benötigen.

Überflüssig sind vor allem Bibliotheken, die von keiner Anwendung (mehr) benötigt werden.

Die Prüfung kann mit

```
apt-cache rdepends <Paketname>
```

erfolgen. Als <Paketname> ist der Name des Binar-Paketes anzugeben.

26.3.2. Fehlerbericht

Für die Durchführung ist nun ein Fehlerbericht zu erstellen. Dieser wird gegen das Pseudopaket *ftp.debian.org* erstellt. ³

¹<https://wiki.debian.org/BSP>

²[9], Abschnitt 5.9-Pakete entfernen

³s. a. [9], Abschnitt 5.9-Pakete entfernen

27. Autopkgtest

28. Lintian-Meldungen

Das Ergebnis des Bauens eines Debian-Paketes wird mit *lintian* überprüft (Kapitel 40.3, Seite 330).

Es gibt ein User's Manual zu *Lintian*, das auch im Paket *lintian* als Datei *lintian.rst* (in englischer Sprache) vorliegt[40].

Hierbei wird der Maintainer oft mit Fehlermeldungen konfrontiert.

Ein vollständiges Verzeichnis der möglichen Meldungen befindet sich unter <https://udd.debian.org/lintian-tag.cgi>.

Eine Liste aller Fehlermeldungen erhält man auch im Terminal:

```
lintian-explain-tags --list-tags | less
```

Die Erläuterung zu einer einzelnen Fehlermeldung erhält man mit

```
lintian-explain-tags <Lintian message>
```

Nachstehend einige Beispiele:

bad-jar-name Der Name entspricht nicht den Richtlinien der Java-Policy.¹

codeless-jar Die *.jar Datei enthält keinen kompilierten Java-Code.

empty-binary-package Das gebaute Paket ist leer.

javailib-but-no-public-jars Im Verzeichnis */usr/share/java/* gibt es kein *.jar-Archiv.

Dann fehlt in der Regel der Eintrag *-java-lib* in der entsprechenden Datei *debian/<paketname.poms*.

new-package-should-close-its-bug In der Datei *debian/changelog* ist der ITP-Bug zu schließen (Closes: #nnnnnn)

rules-requires-root-missing In die Datei *debian/control* wird im ersten Abschnitt der Eintrag *Rules-Requires-Root: no* benötigt.

wildcard-matches-nothing-in-dep5-copyright

backports-changes-missing

out-of-date-standard

testsuite-autopkgtest-missing

Ein Werkzeug zur automatischen Fehlerbehebung ist *lintian-brush*. Dieses Programm befindet sich im gleichnamigen Paket. Allerdings enthält es Skripte für nur etwas mehr als 10 % der möglichen Meldungen.

Mit *lintian-brush --list-tags* kann ermittelt werden, welche Korrekturskripte zur Verfügung stehen.

¹Kapitel 2.4 der Java-Policy[28]

29. Reproduzierbare Builds

29.1. reprotest

30. piuparts

Teil III.

**Wie ein Shell-Skript hilft, ein
Debian-Paket zu bauen**

31. Erste Schritte im Programmskript

Bevor die Arbeit mit dem Programmskript starten kann, werden noch zwei Dinge benötigt:

- ein Account auf <https://salsa.debian.org> und
- ein GPG-Schlüssel zum Signieren der Pakete.

Der Fingerprint des zu verwendenden Schlüssels sollte in der Datei `~/.debian_projekt/fingerprint` abgelegt werden.

Nun geht es mit dem Programmskript los. Dieses hilft beim Bauen eines **Debian**-Paketes und unterstützt das Hochladen desselben. Dabei gibt der Programmablauf eine zweckmäßige Reihenfolge der notwendigen Schritte vor. Das Programmskript baut also keine **Debian**-Pakete, sondern unterstützt als Assistent den Maintainer. Hierauf wird im Eingangsdialog auch ausdrücklich hingewiesen. (Kapitel 31.2, Seite 106)

Voraussetzung ist, dass alle benötigten Pakete installiert und das System entsprechend eingerichtet ist (Kapitel 19, Seite 57).

Das Programmskript ist modular aufgebaut, sodass man an vielen Stellen „aussteigen“ und später wieder „einsteigen“ kann. Man muss also den Bauprozess nicht immer bis zum Abschluss „in einem Zug“ durchführen.

Der für den Nutzer sichtbare Programmablauf beginnt in jedem Fall mit einem Startdialog (Kapitel 31.2, Seite 106).

Das Bauen eines neuen **Debian**-Paketes erfordert zunächst die Anlage eines neuen Projektes (Kapitel 32, Seite 111).

Alle nachfolgende Punkte werden zumindest im Wesentlichen vom Programmskript erledigt.

Konfigurationsdatei Das Programm benötigt eine Konfigurationsdatei (Kapitel 32.1, Seite 111), die zunächst vom Programmskript angelegt wird. Diese kann jederzeit geändert werden.

Systemeinrichtung Um ein **Debian**-Paket zu bauen, müssen unter anderem folgende vorbereitende Aufgaben erfüllt werden:

Bereitstellung der benötigten Verzeichnisse Die Erstellung der Verzeichnisse wird in Kapitel 32.2.1, (Seite 129) beschrieben.

Einrichten eines Git-Repositorys Die Einrichtung des lokalen Git-Repositorys (Kapitel 32.4, Seite 135) muss vor der Ausführung von `gbp import-orig` (Kapitel 34.3.11, Seite 218) erfolgen.

Bereitstellung des Quellcodes Dies wird in Kapitel 34.2, Seite 183 beschrieben.

Bereitstellung der benötigten Dateien im Verzeichnis `debian/` Die Erzeugung der Dateien im Verzeichnis `debian/` erfolgt im Rahmen des Bauens der **Debian**-Revision (Kapitel 35, Seite 225).

Bauen Dies wird in Kapitel 37, Seite 291 beschrieben

Testen – soweit wie möglich Dies wird in Kapitel 40, Seite 327 beschrieben.

Hochladen Dies wird in Kapitel 43, Seite 357 beschrieben.

Bis zum Hochladen ist es ein langer Weg. Doch auch der längste Weg beginnt bekanntlich mit dem ersten Schritt. Dabei ist noch zu beachten, dass das Programmskript tastaturbetont (mit der TAB-Taste) bedient wird. Eine Steuerung mit der Maus kann zu unerwarteten Effekten führen.

31.1. Der Anfang steht am Schluss

Das Programmskript enthält viele Funktionen.

Das Hauptprogramm ruft lediglich die Funktion *BuildApp* auf. Es steht am Ende des Programmskriptes.

```
106a <MainProgram 106a>≡ (164b)
#####
# Here it starts
BuildApp

#####
# This is the end, my friend
```

31.2. Und das sieht der Nutzer als Erstes

Die Funktion *BuildApp* steuert den Programmablauf im Wesentlichen durch den Aufruf weiterer Funktionen.

Sie stellt dem Nutzer als Erstes das Programm vor.

```
106b <BuildApp 106b>≡ (109)
function BuildApp {
    # Called by main program

    #####

    # Intro

    #####

    intro="Assistant to build simple Debian packages\nusing git-buildpackage\n
Authors: Mechtilde Stehmann\n
        Michael Stehmann\n
Version: 0.9.9\n
License: GPL v3+\n
This program does not build Debian packages itself.
It is only an assistant for the package maintainer."

    whiptail --title "Introduction" --msgbox "$intro" 20 60

<BuildApp3 107a>
```

Dazu erscheint der folgende Begrüßungsdialog.

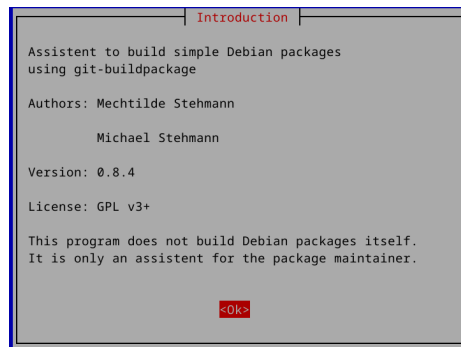


Abbildung 31.1.: Startbildschirm

Danach fragt diese Funktion den Namen des entsprechenden Projektes mit der Funktion *AskOrigName* ab (Kapitel 31.3, Seite 107).

```
107a <BuildApp3 107a>≡ (106b)
    # Definitions of Project
    AskOrigName
    CreateDirsAndLogFile

    # Flag for additional gbp buildpackage options
    OptFlag=0

    #####

    # End of intro
```

<BuildApp7 131a>

Schließlich prüft diese Funktion *BuildApp* das Vorhandensein der notwendigen lokalen Infrastruktur (Konfigurationsdatei (Kapitel 32.1, Seite 111), Verzeichnisse (Kapitel 32.2.1, Seite 129), Git-Repositorium (Kapitel 33.4, Seite 169)) und sorgt gegebenenfalls für deren Anlage.

31.3. Projektname abfragen

Es wird nun der Projektname eines existierenden Projektes abgefragt oder für ein neues Projekt festgelegt. Das Einfügen des Projektname kann auch per *Copy & Paste* erfolgen. Diese Möglichkeit ist eine Besonderheit von *whiptail*.

```
107b <AskOrigName 107b>≡ (177)
    function AskOrigName {
        # Called by BuildApp ConfigFileLEC and itself

        # Name of the project (without this name the app cannot work)
        OrigName=$(whiptail --title "This name is required!" \
            --inputbox "Name of the project:" \
            --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)
    }
    <AskOrigName1 108>
```

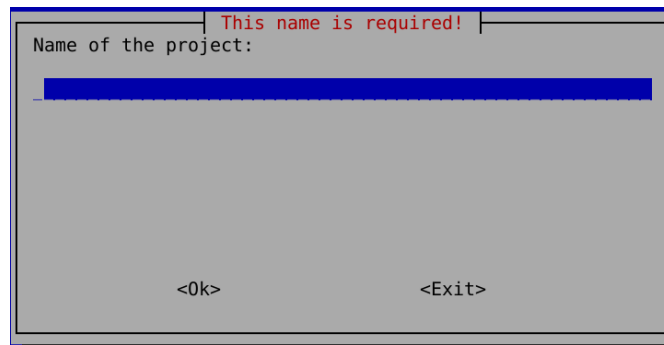


Abbildung 31.2.: Angabe des Projektnamens.

Eine *whiptail --inputbox* erfordert Umleitungen der Ausgabe unter Verwendung von Dateibezeichnern (3>&2 2>&1 1>&3).

```
108 <AskOrigName1 108>≡ (107b)
      if [ $? -ne 0 ]
      then
          whiptail --title "Bye" --msgbox "Bye" 15 60
          exit
      fi
<AskOrigName2 109>
```

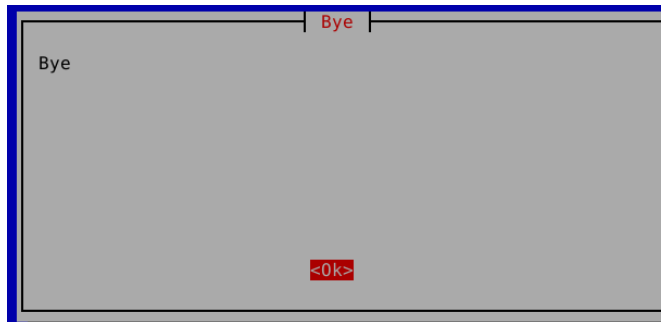



Abbildung 31.3.: Bye

Nur die ersten drei Dateibezeichner (beginnend mit 0) haben eine standardisierte Bedeutung:

- 0 - stdin - Standard-Eingabe
- 1 - stdout - Standard-Ausgabe
- 2 - stderr - Standard-Fehleranzeige

Wenn das Projekt bereits existiert, geht es mit der Anzeige der Konfigurationsdatei (Kapitel 33.1, Seite 165), der Auswahl eines Git-Zweiges (Kapitel 33.4, Seite 169) und dann der Aufgabenauswahl (Kapitel 33.5, Seite 177) weiter.

Wird **kein** Projektname eingegeben, erfolgt ein Hinweis. Die Eingabe eines Projektname ist zwingend erforderlich. Mit *Exit* wird das Programm abgebrochen.

```

109 <AskOrigName2 109>≡ (108)
    if [ -z "${OrigName}" ]
    then
        whiptail --title "No project name" \
        --msgbox "You have to identify a project name\n \
        even it is a new project!" 15 60
        AskOrigName
    fi
    ConfigFileLEC
}

<BuildApp 106b>

```

4. Mai 2025

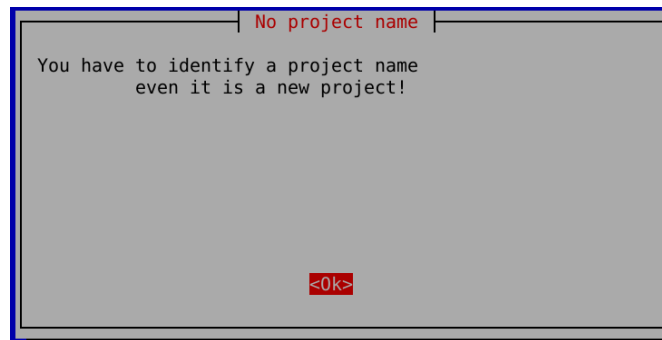


Abbildung 31.4.: Kein Projektname angegeben.

Nach der Bestätigung von *OK* wird erneut der Eingabedialog angezeigt.

31.4. Weiterer Ablauf

Der weitere Ablauf hängt nun davon ab, ob schon eine Konfigurationsdatei für das Projekt, wie in Kapitel 19.2.2.1, Seite 59 beschrieben, existiert.

In diesem Fall wird diese „geladen“ (Kapitel 33.1, Seite 165).

Andernfalls wird die Konfigurationsdatei neu angelegt.

32. Anlegen eines neuen Projektes

Zur Anlage eines neuen Projektes werden zunächst die Konfigurationsdatei erstellt und die notwendige Infrastruktur angelegt.

32.1. Konfigurationsdatei erstellen

Die Konfigurationsdatei wird im Homeverzeichnis des Nutzers im Verzeichnis *.debian_project/* als Datei *<Projektname>* gespeichert.

```
111a <ConfigFileLEC 111a>≡ (167a)
function ConfigFileLEC {
    # Called by AskOrigName CreateNewBranch TaskSelect OwnServer

    ## Load, edit or create config file - using AskConfig

    # Path to config files directory
    ConfigPath=~/.debian_project/
    changeflag=0
```

<ConfigFileLEC1 165a>

Die Funktion *ConfigFileLEC* prüft zunächst, ob eine Konfigurationsdatei mit dem Projektnamen vorhanden ist. Ist das Ergebnis der Prüfung negativ, erfolgt die Meldung, dass keine Konfigurationsdatei mit diesem Namen gefunden werden konnte.

```
111b <ConfigFileLEC4 111b>≡ (166b)
    else
        if whiptail --title "Config file not found" \
            --yesno "There is no config file for ${OrigName}\n \
            which you can edit.\n \
            Do you want to create a new project?" \
            --yes-button "Yes" --no-button "No" 15 60
        then
            changeflag=1
```

<ConfigFileLEC5 112>

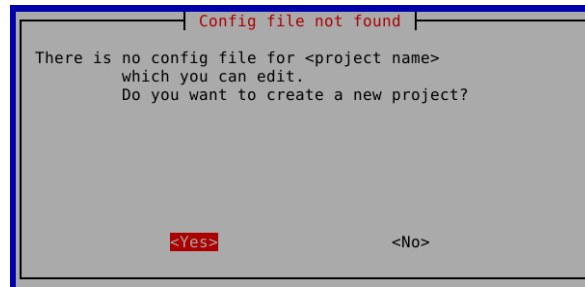


Abbildung 32.1.: Keine Konfigurationsdatei gefunden.

Ein *whiptail -yesno* gibt 0 bzw true zurück, wenn die Frage bejaht wird, und 1 bzw false, wenn sie verneint wird.

Wird diese Frage bejaht, wird vorsorglich auch das Verzeichnis *.debian_project* als *ConfigPath*, sofern noch nicht vorhanden, angelegt. Dann wird die Funktion *AskConfig* aufgerufen, mit der die Konfigurationsdatei erstellt wird.

```
112 <ConfigFileLEC5 112>≡ (111b)
      mkdir --parents ${ConfigPath}
      AskConfig
    else
      AskOrigName
    fi
  fi
  set -e
}

<CreateDirsAndLogfile 129>
```

Die Abfrage, ob ein neues Projekt erstellt werden soll, ermöglicht dem Nutzer, Tippfehler bei der Eingabe des Projektnamens zu korrigieren, wenn er sie verneint.

In den folgenden Abschnitten werden die in der Konfigurationsdatei enthaltenen Variablen besprochen. Zunächst folgen die Variablen, die für alle Pakete erforderlich sind. Danach folgen die Variablen, die nur für jeweils eine Gruppe von Paketen benötigt werden (für Java-Pakete s. Kapitel 32.1.2.2, Seite 123, für *webevt*-Pakete s. Kapitel 32.1.2.3, Seite 125).

32.1.1. Abfrage allgemeiner Variablen für die Konfigurationsdatei

Die Funktion *AskConfig* ordnet in ihrem ersten Teil den Variablen, die in der Konfigurationsdatei enthalten sein sollen, Werte zu. Dies geschieht durch die Abfrage der einzelnen Variablen. Das Speichern der Konfigurationsdatei wird dann in Kapitel 32.1.3, Seite 127 beschrieben.

Es wird geprüft, ob der jeweiligen Variablen ein Wert zugewiesen wurde.

Dies geschieht als Erstes für den Namen des Quellpaketes, welcher als Wert der Variablen *SourceName* zugewiesen wird.

Der Name des Quellpaketes ist der Name, den das Upstream-Projekt seiner Software gegeben hat.

```

113 <AskConfig 113>≡ (123a)
    function AskConfig {
        # Called by ConfigFileLEC

        if [ -z "${SourceName}" ]
        then
            SourceName=$(whiptail --title "Source Package Name" \
                --inputbox "Name of the source package:" \
                --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)

            if [ $? -ne 0 ] # Cancel-Button was pressed
            then
                exit
            else
                changeflag=1
            fi
        fi

        <AskConfig1 114a>

```

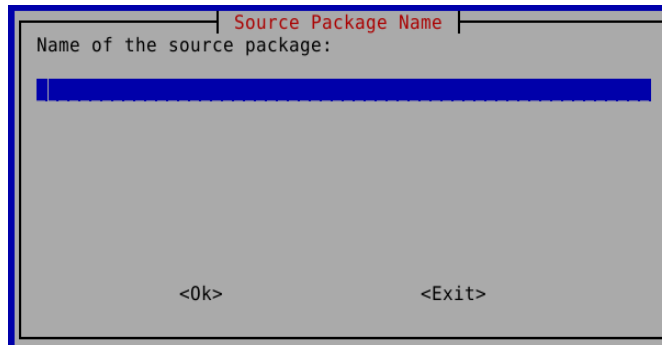


Abbildung 32.2.: Name des Quellcode-Paketes

Enthält die Variable einen Wert, wird gefragt, ob dies der richtige Wert sei. Dies dient zum Einen der Kontrolle der vorherigen Eingabe und zum Anderen eröffnet dies die Möglichkeit des Editierens (Kapitel 33.1, Seite 165) der Konfigurationsdatei mittels der Funktion *AskConfig*.

```
114a <AskConfig1 114a>≡ (113)
      if ! whiptail --title "Source Package Name" \
        --yesno "The name of the source package is ${SourceName}" \
        --yes-button "Yes" --no-button "No" 15 60
      <AskConfig2 114b>
```

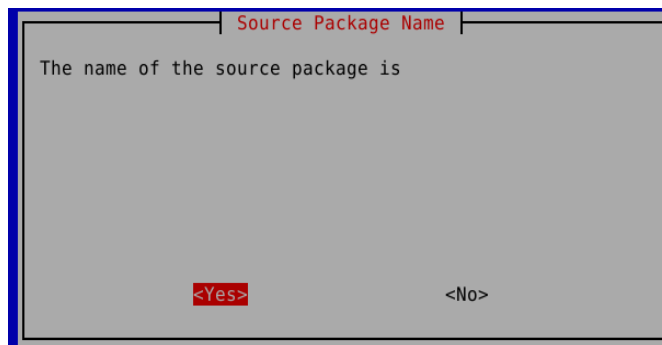


Abbildung 32.3.: Namen des Quellpaketes angeben

```
114b <AskConfig2 114b>≡ (114a)
      then
        SourceName=$(whiptail --title "Name of the source package" \
          --inputbox "Real name of the source package:" \
          --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)

        if [ ${#SourceName} -eq 0 -o $? -ne 0 ]
        then
          exit
        else
          changeflag=1
        fi
      fi

      <AskConfig3 115>
```

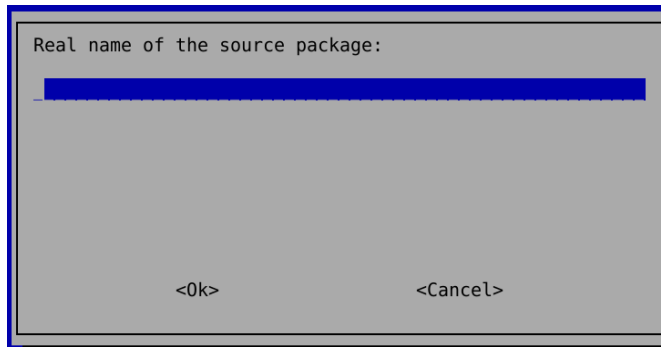


Abbildung 32.4.: Korrekten Namen des Quellpaketes angeben

Nun wird der Name des zu bauenden Paketes ermittelt. Dafür wird der Variablen *PackName*, wenn sie noch nicht existiert, der Wert der Variablen *SourceName* als Standardwert zugewiesen. Auch hierzu wird eine Bestätigung abgefragt und eine Korrekturmöglichkeit eröffnet.

```

115  <AskConfig3 115>≡ (114b)
      if [ -z "${PackName}" ]
      then
          PackName=${SourceName}
          tadd=", too?"
      else
          tadd="?"
      fi

      if ! whiptail --title "PackName" \
          --yesno "The name of the package is ${PackName}${tadd}" \
          --yes-button "Yes" --no-button "No" 15 60
      <AskConfig3-1 116>

```

4. Mai 2025

Wenn die Variable *PackName* noch nicht existierte:

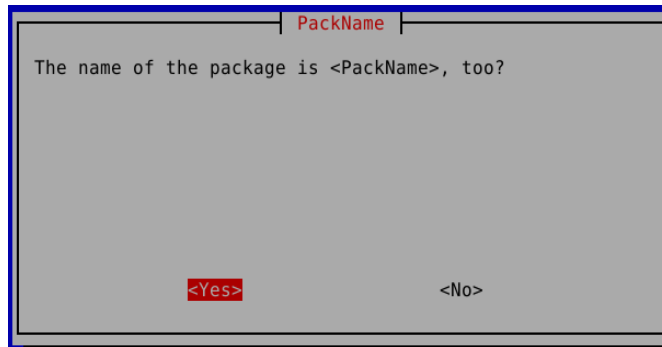


Abbildung 32.5.: Ist der Paketname korrekt?

Wenn die Variable *PackName* bereits existierte:

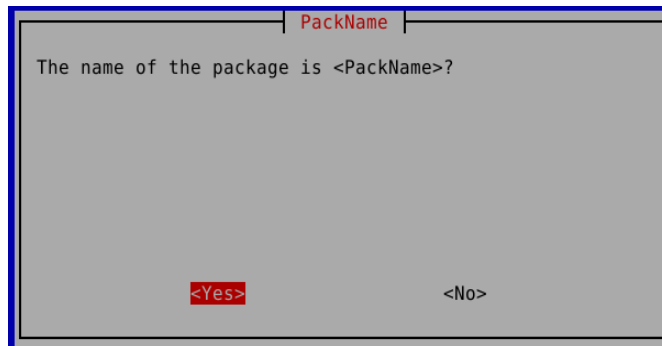


Abbildung 32.6.: Ist der Paketname korrekt?

```
116 <AskConfig3-1 116>≡ (115)
    then
      PackName=$(whiptail --title "Name of the Debian Package" \
        --inputbox "Real name of the package,\nwhich should be built from ${SourceName}:" \
        --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)

      if [ ${#PackName} -eq 0 -o $? -ne 0 ]
      then
        exit
      else
        changeflag=1
      fi
    fi

<AskConfig4 117>
```

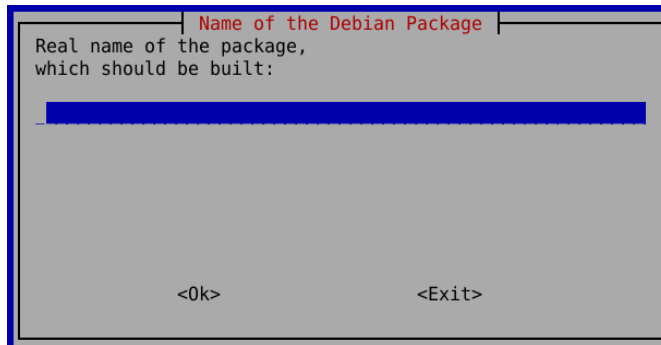



Abbildung 32.7.: Korrekten Name des Paketes angeben.

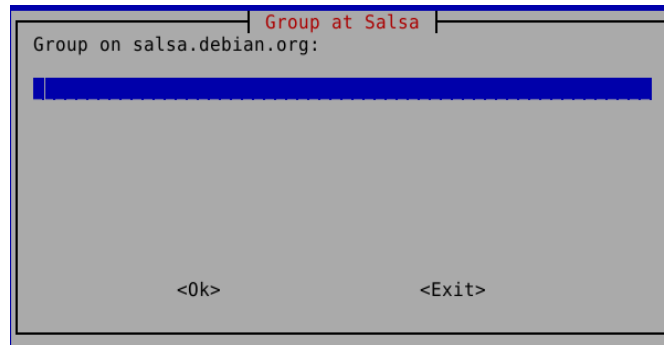
Nun wird der Name der Gruppe abgefragt, unter welcher das Git-Repositorium auf *salsa.debian.org* angelegt werden soll.

```

117 <AskConfig4 117>≡ (116)
    if [ -z "${SalsaName}" ]
    then
        SalsaName=$(whiptail --title "Group at Salsa" \
        --inputbox "Group on salsa.debian.org:" \
        --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)
        if [ $? -ne 0 ] # Cancel-Button was pressed
        then
            exit
        else
            SalsaName="${SalsaName}/${SourceName}.git"
        fi
    fi

```

<AskConfig5 118a>

Abbildung 32.8.: Name der Gruppe auf *salsa.debian.org* angeben.

Es gibt verschiedene Paketier-Teams (beispielsweise für Java- oder Python-Pakete)¹. Beim Python-Team muss zusätzlich das Verzeichnis *packages* angegeben werden, also *python-team/packages*.

Diese Teams verfügen über eigene Gruppen auf *salsa.debian.org*. Mit <https://salsa.debian.org/explore/groups> können alle öffentlichen Gruppen angezeigt werden.

Soll ein Paket unabhängig von einem Paketier-Team betreut werden, ist als Gruppe *Debian* einzutragen.

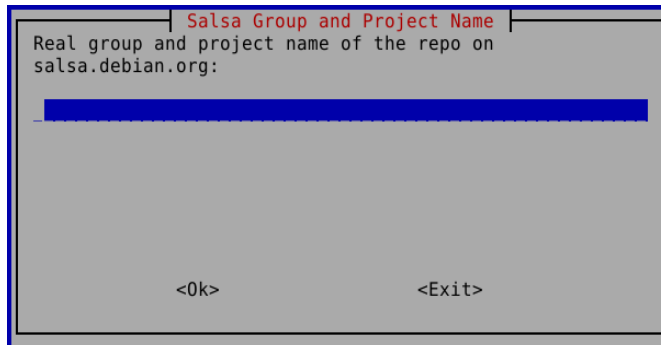
```
118a <AskConfig5 118a>≡ (117)
      if ! whiptail --title "Salsa Name" \
        --yesno "Group and project name of the repo on salsa.debian.org is $SalsaName" \
        --yes-button "Yes" --no-button "No" 15 60
      <AskConfig5-1 118b>
```

Abbildung 32.9.: Name der Gruppe auf *salsa.debian.org* angeben.

```
118b <AskConfig5-1 118b>≡ (118a)
      then
        SalsaName=$(whiptail --title "Salsa Group and Project Name" \
          --inputbox "Real group and project name of the repo on salsa.debian.org:" \
          --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)

      <AskConfig5-2 119a>
```

¹https://wiki.debian.org/Teams#Packaging_teams

Abbildung 32.10.: Name der Gruppe auf *salsa.debian.org* angeben.

```
119a <AskConfig5-2 119a>≡ (118b)
      if [ $#SalsaName -eq 0 -o $? -ne 0 ]
      then
          exit
      else
          changeflag=1
      fi
fi
```

<AskConfig6 119b>

Es wird nun geprüft, ob im Verzeichnis *~/.debian_project* eine Datei *DefaultValues* existiert. In dieser Datei werden Variablen Werte zugewiesen, die für viele Projekte gelten (Kapitel 19.2.2.2, Seite 60). Das Skript *DefaultValues* wird dann ausgeführt.

```
119b <AskConfig6 119b>≡ (119a)
      if [ -f ${ConfigPath}/DefaultValues ]
      then
          . ${ConfigPath}/DefaultValues
      fi
```

<AskConfig7 119c>

Der Variablen *DefaultProjectPath* ist als Wert der Pfad zugewiesen, welcher zu dem Verzeichnis führt, das die einzelnen Projektverzeichnisse als Unterverzeichnisse enthält.

```
119c <AskConfig7 119c>≡ (119b)
      if [ -n "${DefaultProjectPath}" ]
      then
          ProjectPath=${DefaultProjectPath}
      fi

      if [ -z "${ProjectPath}" ]
      then
          ProjectPath=$(whiptail --title "Path to Project Directory" \
            --inputbox "Path to the project directory on your local machine\n \
            (without '/${OrigName}':" \
            --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
          <AskConfig8 120a>
```

4. Mai 2025

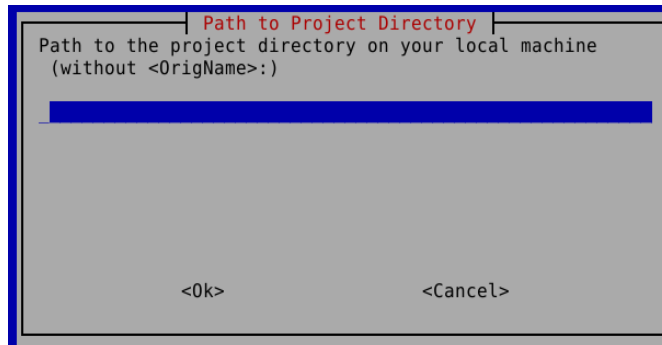


Abbildung 32.11.: Pfad zum Projektverzeichnis auf der lokalen Maschine

120a $\langle AskConfig8\ 120a \rangle \equiv$

(119c)

```
if [ -z "${ProjectPath}" ]
then
    echo -e "Path to the project directory on your local machine\n \
    (without '${OrigName}':)"
    read ProjectPath
fi
changeFlag=1
fi

if ! whiptail --title "ProjectPath" \
--yesno "Path to the project directory on your local machine \
is ${ProjectPath}/${OrigName}" \
--yes-button "Yes" --no-button "No" 15 60
```

$\langle AskConfig9\ 120b \rangle$

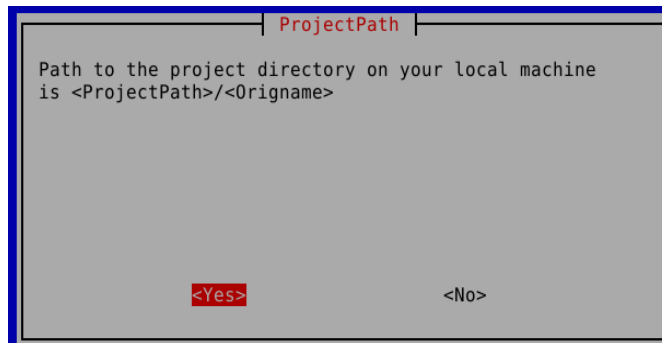


Abbildung 32.12.: Pfad zum Projektverzeichnis auf der lokalen Maschine mit OrigName

120b $\langle AskConfig9\ 120b \rangle \equiv$

(120a)

```
then
    ProjectPath=$(whiptail --title "Path to Project Directory" \
--inputbox "Real path to the project directory on your local machine\n \
(without '${OrigName}':)" \
--cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
 $\langle AskConfig9-1\ 121 \rangle$ 
```

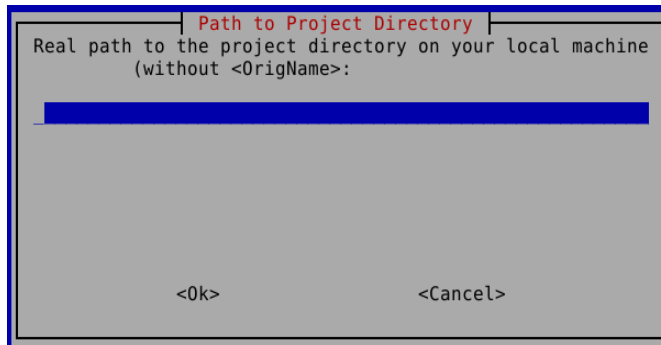


Abbildung 32.13.: Pfad zum Projektverzeichnis auf der lokalen Maschine (real)

```

121  <AskConfig9-1 121>≡ (120b)
      if [ -z "${ProjectPath}" ]
      then
          echo -e "Path to the project directory on your local machine\n \
          (without '${OrigName}':"
          read ProjectPath
      fi
      if [ -z "${ProjectPath}" ]
      then
          exit
      else
          changeflag=1
      fi
  fi

```

<AskConfig10 123b>

32.1.2. Abfrage spezieller Variablen für die Konfigurationsdatei

Nun werden die Variablen ermittelt, die nur für spezielle Arten von Paketen benötigt werden. Begonnen wird mit den Variablen für Java-Paketen (Kapitel 32.1.2.2, Seite 123)

Außerdem wird gegebenenfalls in die Konfigurationsdatei der Pfad zu einem benötigten Plugin eingetragen.

32.1.2.1. Ermittlung der Plugin-Pfade

Bestimmte Arten von Paketen werden unter Verwendung von Plugins zum (Haupt-)Skript gebaut. Es handelt sich hierbei um Pakete, die mit Java, *maven* gebaut werden, um Mozilla-Erweiterungen und Python-Programme.

Die Funktion *DetectPlugins* ermittelt die Pfade zu Plugin-Skriptdateien. Sie kann für weitere Plugins verwendet werden. Sie wird derzeit nicht benutzt.

Beim Aufruf von *DetectPlugins* müssen zwei Optionen übergeben werden. Dabei wird die erste Option (*\$1*) an *PluginName* und die zweite (*\$2*) an *PluginFile* übergeben.

```

122 <DetectPlugins 122>≡ (160)
    function DetectPlugins {
        # Not Used.
        # Funktion to find Plugins

        # This function needs two options: the name of the plugin
        # and the name of the plugin script file
        PluginName=$1
        PluginFile=$2

        # Determine path to the (needed) plugin script
        PluginPathL=$(locate ${PluginFile})
        PluginPathA=(${PluginPathL})

        # If there is only one result
        if [ ${#PluginPathA[@]} -eq 1 ]
        then
            PluginPath=${PluginPathA[0]}
        # If there are some results
        elif [ ${#PluginPathA[@]} -gt 1 ]
        then
            i=0; slct=''
            for element in ${PluginPathA[*]}
            do
                slct=$slct' '$i' '${element}' off '
                i=$(expr $i + 1)
            done

            PluginNr=$(whiptail --title "${PluginName} plugin found" \
                --radiolist "Select:" 15 60 5 $slct \
                --cancel-button "Cancel" 3>&2 2>&1 1>&3)

    <DetectPlugins3 123a>

```

```

123a  <DetectPlugins3 123a>≡ (122)
      if [ $? -eq 1 ]
      then
          return 24
      fi

      PluginPath=${PluginPathA[${PluginNr}]}
      # If there is no result
      else
          PluginFlag=0
          whiptail --title "File not found" \
          --msgbox "'${PluginFile}' was not located!" \
          15 60
          return 24
      fi
  }

  <AskConfig 113>

```

32.1.2.2. Variablenabfrage für Java-Pakete

Bestimmte Arten von Paketen erfordern eine spezielle Behandlung. Diese erfolgt unter anderem durch separate Skripte. Diese Behandlung wird durch „flags“ gesteuert. Damit wird auch ein unnötiges Laden der Plugin-Skripte vermieden.

```

123b  <AskConfig10 123b>≡ (121)
      # Java Flag
      # This is needed to trigger special entries
      # in debian/control and debian/rules
      if [ -z "${JavaFlag}" ]
      then
          if whiptail --title "Java" --defaultno \
          --yesno "Do you want to build a Java package?" \
          --yes-button "Yes" --no-button "No" --defaultno 15 60

```

<AskConfig10-1 124a>

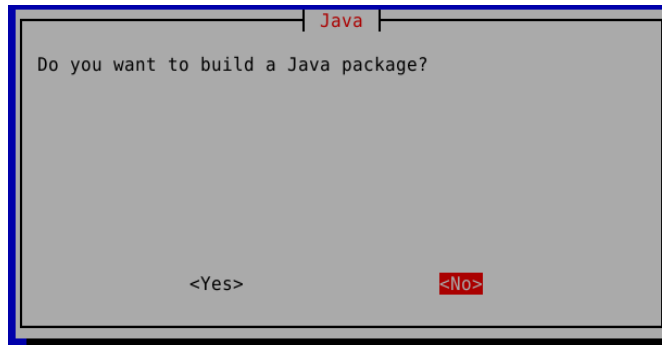


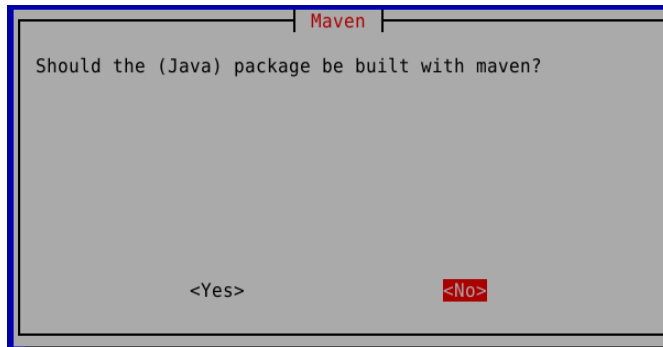
Abbildung 32.14.: Soll ein Java-Paket gebaut werden?

```
124a <AskConfig10-1 124a>≡ (123b)
    then
        JavaFlag=1
    else
        JavaFlag=0
    fi
    changeflag=1
fi
```

<AskConfig11 124b>

Abgefragt wird auch, ob das Maven-Plugin (Kapitel 47, Seite 385) verwendet werden soll. Dieses wird dann im System lokalisiert und der Pfad zum Skript in der Konfigurationsdatei hinterlegt.

```
124b <AskConfig11 124b>≡ (124a)
    # Maven Plugin
    if [ ${JavaFlag} -eq 1 ]
    then
        if [ -z "${MavenPluginFlag}" ]
        then
            if whiptail --title "Maven" --defaultno \
                --yesno "Should the (Java) package be built with maven?" \
                --yes-button "Yes" --no-button "No" --defaultno 15 60
            <AskConfig12 125a>
```


Abbildung 32.15.: Soll ein Java-Paket mit *maven* gebaut werden?125a `<AskConfig12 125a>≡`

(124b)

```

    then
        MavenPluginFlag=1
    else
        MavenPluginFlag=0
    fi
    changeflag=1
fi
fi

```

`<AskConfig15 125b>`

Beim Aufruf von *DetectPlugins* müssen zwei Optionen übergeben werden. Dabei wird 'Maven' als *\$1* an die Variable *PluginName* und *build-gbp-maven-plugin.sh* als *\$2* an die Variable *PluginFile* übergeben (s. Kapitel 32.1.2.1, Seite 122).

32.1.2.3. Variablenabfrage für Mozilla-Erweiterungen

125b `<AskConfig15 125b>≡`

(125a)

```

# Webext Flag
# This is needed to trigger special entries
# in debian/control and debian/rules
if [ -z "${WebextFlag}" ]
then
    if whiptail --title "Mozilla AddOns" --defaultno \
        --yesno "Do you want to build a Mozilla AddOn package?" \
        --yes-button "Yes" --no-button "No" --defaultno 15 60
    <AskConfig15-1 126a>

```

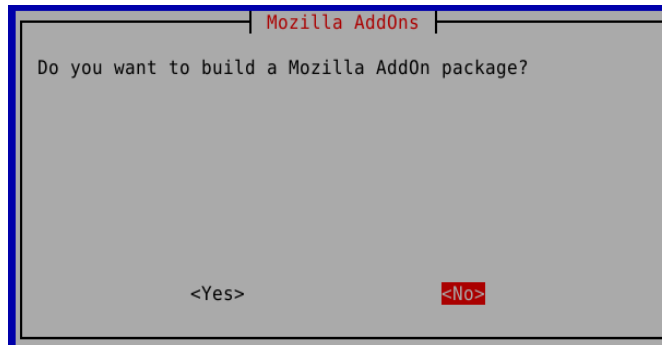


Abbildung 32.16.: Soll eine Erweiterung für Mozilla paketiert werden?.

```
126a <AskConfig15-1 126a>≡ (125b)
      then
        WebextFlag=1
      else
        WebextFlag=0
      fi
      changeflag=1
    fi
```

<AskConfig18 126b>

32.1.2.4. Variablenabfrage für Python3-Pakete

Auch für das Paketieren von Python-Paketen kann auf ein entsprechendes Plugin zurückgegriffen werden. Dafür werden zunächst die benötigten Informationen abgefragt und in die Konfigurationsdatei eingetragen.

```
126b <AskConfig18 126b>≡ (126a)
      # Python Flag
      # This is needed to trigger special entries
      # in debian/control and debian/rules
      if [ -z "${PythonFlag}" ]
      then
        if whiptail --title "Python3 programs" --defaultno \
          --yesno "Do you want to build a Python3 package?" \
          --yes-button "Yes" --no-button "No" --defaultno 15 60
        <AskConfig19 127a>
```

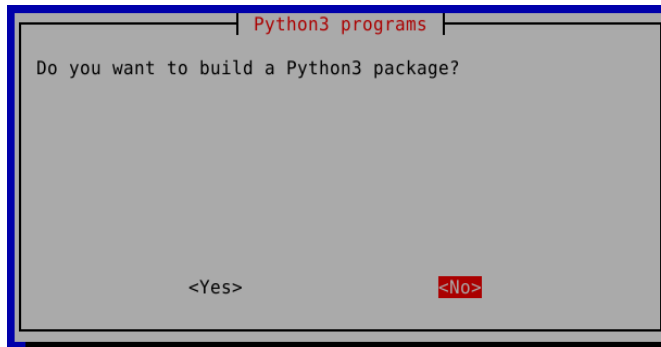


Abbildung 32.17.: Soll ein Python3-Paket gebaut werden?.

```
127a  <AskConfig19 127a>≡ (126b)
      then
        PythonFlag=1
      else
        PythonFlag=0
      fi
      changeflag=1
    fi
```

<AskConfig20 127b>

32.1.3. Speichern der Konfiguration

Das (Neu-)Erstellen der Konfigurationsdatei geschieht im folgenden Teil der Funktion *AskConfig*.

Es wird zunächst geprüft, ob bereits eine Konfigurationsdatei für das Projekt existiert. Sofern vorhanden wird sie gelöscht.

Die Konfigurationsdatei ist ein Shell-Skript. Sie beginnt daher mit einer entsprechenden *Shebang*.

```
127b  <AskConfig20 127b>≡ (127a)
      if [ $changeflag -eq 1 ]
      then
        if [ -f ${ConfigPath}${OrigName} ]
        then
          rm ${ConfigPath}${OrigName}
        fi
        touch ${ConfigPath}${OrigName}

        # Shebang of the config file
        SB='#!/usr/bin/bash'

        # The config file is a shell script
        echo ${SB} >> ${ConfigPath}${OrigName}
        echo '# ConfigFile for '${OrigName}' >> ${ConfigPath}${OrigName}
        echo '## General parameters' >> ${ConfigPath}${OrigName}
        echo 'SourceName='${SourceName}' >> ${ConfigPath}${OrigName}
        echo 'PackName='${PackName}' >> ${ConfigPath}${OrigName}
        echo 'ProjectPath='${ProjectPath}' >> ${ConfigPath}${OrigName}
        echo 'SalsaName='${SalsaName}' >> ${ConfigPath}${OrigName}
```

<AskConfig22 128a>

4. Mai 2025

Die folgenden Einträge in der Konfigurationsdatei werden nur erstellt, wenn entsprechende Flags gesetzt wurden. Dies gilt für Java-Pakete. Ein zusätzlicher Eintrag erfolgt, wenn sie mit *maven* gebaut werden. Dies gilt auch für die Erweiterungen für Firefox und Thunderbird und Programme in der Programmiersprache Python.

```
128a <AskConfig22 128a>≡ (127b)
    echo '## Parameters for Java packages'>> ${ConfigPath}${OrigName}
    echo 'JavaFlag=${JavaFlag} >> ${ConfigPath}${OrigName}
    if [ ${JavaFlag} -eq 1 ]
    then
        echo 'MavenPluginFlag=${MavenPluginFlag} >> ${ConfigPath}${OrigName}
    fi

    echo '## Parameters for Webext packages'>> ${ConfigPath}${OrigName}
    echo 'WebextFlag=${WebextFlag} >> ${ConfigPath}${OrigName}

    echo '## Parameters for Python3 packages'>> ${ConfigPath}${OrigName}
    echo 'PythonFlag=${PythonFlag} >> ${ConfigPath}${OrigName}s
    changeflag=0
    fi
}
```

<ReplaceTilde 128b>

Da beim Ablauf des Skriptes die Tilde (~) im Pfad nicht automatisch durch */home/<username>* ersetzt wird, muss dies durch eine Funktion *ReplaceTilde* im Programmskript geschehen. Ferner wird ein eventueller Slash (/) am Ende des Pfades entfernt.

```
128b <ReplaceTilde 128b>≡ (128a)
function ReplaceTilde {
    # Called by ConfigFileLEC GbpConfIntegration
    RecentUser=$(whoami)
    set +e
    tp=$(echo ${SuspectPath} | grep --count '~')
    if [ $tp -ge 1 ]
    then
        CleanPath=$(echo ${SuspectPath} | \
            sed --expression="s/~~/\home/${RecentUser}/g")
    else
        CleanPath=${SuspectPath}
    fi

    # Replace / at the end
    CleanPath=$(echo ${CleanPath} | sed --expression="s/\$///")
    set -e
}
```

<SetDefaults 167a>

32.1.4. Beispiel einer Konfigurationsdatei

```
#!/usr/bin/bash
# ConfigFile for <OrigName>
## General parameters
SourceName=<SourceName>
PackName=<PackName>
ProjectPath=/home/mechtilde/Projekte/Git/01_Salsa
SalsaName=<Name of the team>/<SourceName>.git
## Parameters for Java packages
JavaFlag=0
## Parameters for Webext packages
WebextFlag=0
## Parameters for Python3 packages
PythonFlag=0
RecentBranch=debian/sid
## Maintainer and Uploaders
Maintainer=<Name and E-Mail-Address of the Maintainer>
Uploaders=<Name and E-Mail-Address of the Uploaders>
## Download from upstream
DownloadURL=<Upstream URL for download>
RecentUpstreamSuffix=.xpi
# debian/sid_Dist=sid
```

32.2. Anlegen der Infrastruktur

Zur Infrastruktur jedes Projektes gehören Verzeichnisse, eine Log-Datei und ein Git-Repository. Sofern diese nicht bereits vorhanden sind, werden die notwendigen Verzeichnisse und die Log-Datei angelegt.

Zur Infrastruktur gehören auch ein *Chroot*-Verzeichnis *base.cow* oder ein *Sbuild-Chroot*-Verzeichnis. Diese werden nur angelegt, wenn sie für die Distribution, für die das Paket gebaut werden soll, noch nicht existieren.

32.2.1. Anlegen der notwendigen Verzeichnisse

Zunächst werden die zuvor definierten Pfade (Kapitel 32.2.2, Seite 130) angelegt.

```
129 <CreateDirsAndLogfile 129>≡ (112)
function CreateDirsAndLogFile {
    # Called by BuildApp ConfigFileLEC

    # Replace tilde if necessary
    SuspectPath=${ProjectPath}
    ReplaceTilde
    ProjectPath=${CleanPath}

<CreateDirsAndLogfile1 130a>
```

32.2.2. Definition der Pfade

Am Ende der Funktion *ConfigFileLEC* werden noch zwei zusammengesetzte Pfade definiert. Die Beschreibung dieser Pfade findet sich in Kapitel 19.2 (Seite 58).

```
130a <CreateDirsAndLogfile1 130a>≡ (129)
      # Set paths
      PrjPath=${ProjectPath}/${OrigName}
      GitPath=${PrjPath}/${SourceName}

      # Create directories if necessary
      mkdir --parents ${PrjPath} # redundantly?
      mkdir --parents ${GitPath}

      <CreateLogFile 130b>
```

32.2.3. Anlegen der Log-Datei

In die Log-Datei werden bei jedem Programmstart zunächst Datum und Uhrzeit eingetragen.

```
130b <CreateLogFile 130b>≡ (130a)
      # Create Log-File
      cd ${PrjPath}
      log=${PrjPath}/${OrigName}.log.txt
      touch ${log}
      echo -e "\n\n===\n=== $(date) ===\n===\n">>${log}
      echo "ConfigFile OK!" >> ${log}
    }

      <InsertDebName 145b>
```

32.3. Git-Repositoryen

Für die Arbeit mit *git-buildpackage* ist ein lokales Git-Repository mit Arbeitsverzeichnis wesentlich.

Das Programmskript geht davon, dass sowohl ein lokales Repository (Kapitel 20.4, Seite 74) als auch ein Repository auf *salsa.debian.org* (Kapitel 21, Seite 75) angelegt werden soll. Daneben berücksichtigt das Programmskript auch ein Git-Repository auf einem eigenen Git-Server (Kapitel 20.4.2, Seite 74).

Die Neuanlage eines Git-Repositorys ist der erste Schritt zum Bauen eines neuen Debian-Paketes. Danach wird der Quellcode für die (neue) Version heruntergeladen (Kapitel 34.2, Seite 183), eine Revision gebaut (Kapitel 35, Seite 225) und schließlich hochgeladen. (Kapitel 43, Seite 357). Daher wird der Nutzer gefragt, ob er ein neues Paket bauen will.

32.3.1. Gibt es bereits ein Git-Repository?

Vorsorglich wird geprüft, ob bereits ein lokales Git-Repository für das anzulegende Projekt existiert. Es wird berücksichtigt, wenn in einem übergeordneten Verzeichnis ein Git-Repository existiert.

Existiert ein lokales Git-Repository, geht es mit der Auswahl eines Git-Zweiges weiter (s. Kapitel 33.4, Seite 169).

```
131a  <BuildApp7 131a>≡ (107a)
      ## Checks whether there is a git repo
      cd ${GitPath}
      set +e
      git status 1>/dev/null 2>&1

      # '==' does the same as '-eq'
      if [ $? == 0 ]
      then
          if [ -d .git ]
          then
              SelectBranch
          else
              <BuildApp8 131b>
```

Wenn ein Git-Repository in einem übergeordneten Verzeichnis existiert, wird ein entsprechender Hinweis gegeben.

```
131b  <BuildApp8 131b>≡ (131a)
      SupOrdMsg="Is there a git repository in a superordinate directory?"
      echo ${SupOrdMsg} >> ${log}
      whiptail --title="Attention!" --msgbox "${SupOrdMsg}" 15 60
      StartTasks
      fi
      else
      StartTasks
      fi

      <BuildApp10 164b>
```

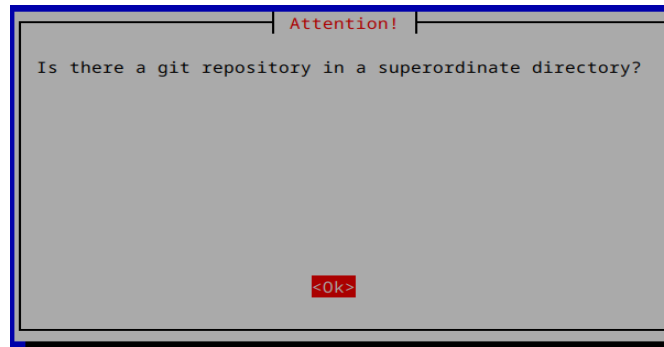


Abbildung 32.18.: Gibt es ein übergeordnetes Git-Repository?

Existiert kein Git-Repository im entsprechenden Verzeichnis, wird die Funktion *StartTasks* aufgerufen.

32.3.2. Auswahldialog

Existiert noch kein Git-Repository, stellt das Skript vier Möglichkeiten bereit, ein solches Repository zu erstellen. Dies sind die Neuanlage mittels *git init* und das Klonen eines bestehenden (Git-)Repositorys von *salsa.debian.org* (Kapitel 32.5, Seite 150). Letzteres setzt vor allem voraus, dass dort ein Zweig *pristine-tar* vorhanden ist.

Andernfalls kann ein Repository durch *gbp import-dsc* angelegt werden (Kapitel 32.6, Seite 156).

Schließlich wird auch der Sonderfall des Sponsorings berücksichtigt (Kapitel 32.7, Seite 158)

```
132 <StartTasks 132>≡
    function StartTasks {
        # Called by BuildApp

        Task=$(whiptail --title "Tasks for building a new package:" \
        --radiolist "What do you like to do to build a new package?" 17 60 9 \
        "0" "Create a git repo and download upstream code" on \
        "11" "Clone an existing repo from Salsa" off \
        "12" "Importing already existing Debian packages" off \
        "13" "Importing from mentors.debian.net for sponsoring" off \
        --cancel-button "Exit" 3>&2 2>&1 1>&3)

        <StartTasks1 133a>
```

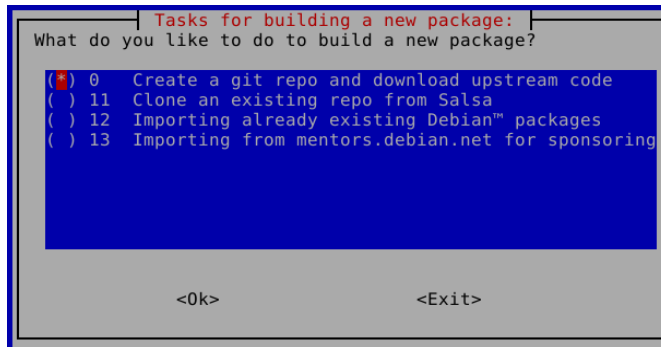



Abbildung 32.19.: Ein neues Paket erstellen

133a `<StartTasks1 133a>≡`

(132)

```

if [ -z "${Task}" ]
then
    exit
fi
TaskSelect
}

```

`<DebDiffCheckList 336b>`

Die Punkte des Menüs befassen sich mit verschiedenen Wegen, ein neues Projekt aufzusetzen. Es geht vor allem um die Beschaffung des Quellcodes des Upstream-Projektes.

Der Aufruf der entsprechenden Funktionen erfolgt durch die Funktion *TaskSelect*.

Bei *Create a git repo and download upstream code* wird durch die Funktion *BuildNewPackage* zunächst das lokale Git-Repositorium angelegt (Kapitel 32.4, Seite 135). Sodann wird eine neue Version heruntergeladen (Kapitel 34.2, Seite 183).

133b `<TaskSelect 133b>≡`

```

function TaskSelect {
    # Called by StartTasks CommonTasks

    ## The lines below serve also the sequence control. A called function
    ## changes finally the variable 'Task', so one of the following
    ## if-clauses matches. That is why the following lines can not be
    ## replaced by a case statement.

    NoPull=0 # Flag for PullFromSalsa

    ## Start tasks

    # Building a new package from archive
    if [ $Task -eq 0 ]
    then
        BuildNewPackage
    fi
}

```

`<TaskSelect0 134a>`

Bei *Clone an existing repo from Salsa* wird ein bereits vorhandenes Repository von *salsa.debian.org* heruntergeladen. Die Funktion *CloneFromSalsa* kloniert dieses Repository mittels des Befehles *gbg clone*. (Kapitel 32.5, Seite 150)

134a `<TaskSelect0 134a>≡` (133b)

```
# Clone an existing repo from Salsa
if [ $Task -eq 11 ]
then
    CloneFromSalsa
fi
<TaskSelect1 134b>
```

Bei *Importing already existing Debian package* wird eine **.dsc*-Datei heruntergeladen und durch das Programm *gbp import-dsc* ein Git-Repository angelegt (Kapitel 32.6, Seite 156).

134b `<TaskSelect1 134b>≡` (134a)

```
# Importing already existing Debian package
if [ $Task -eq 12 ]
then
    ImportDebianPackage
fi
<TaskSelect2 134c>
```

Ein spezieller Fall für die Erstellung eines Git-Repositorys mit *gbp import-dsc* ist das Sponsoring (Kapitel 32.7, Seite 158). Dabei unterstützt ein erfahrenes Mitglied des Debian-Projektes einen Maintainer ohne Upload-Rechte, indem das Mitglied das Paket signiert und hochlädt.

134c `<TaskSelect2 134c>≡` (134b)

```
# Importing from mentors.debian.net for sponsoring
if [ $Task -eq 13 ]
then
    Import4Sponsoring
fi
```

```
<TaskSelect3 178>
```

32.4. Neuanlage eines lokalen Git-Repositorys

Eine Möglichkeit der Erstellung eines Git-Repositorys ist die Neuanlage mittels *git init*. Dies erfolgt in der Funktion *BuildNewPackage*. Zuvor erfolgt noch ein entsprechender Eintrag in der Log-Datei.

```

135a  <BuildNewPackage 135a>≡ (149b)
      function BuildNewPackage {
        # Called by TaskSelect
        cd ${GitPath}
        if [ -d .git ]
        then
          echo "There seems already to be a git repository in ${GitPath}." >> ${log}
          if ! whiptail --title "Warning" \
            --yesno "There seems already to be a git repository in ${GitPath}.\n \
            However continue?" --yes-button "Yes" --no-button "No" 15 60
          then
            echo "Exit" >> ${log}
            exit
          fi
        <BuildNewPackage1 135b>

```

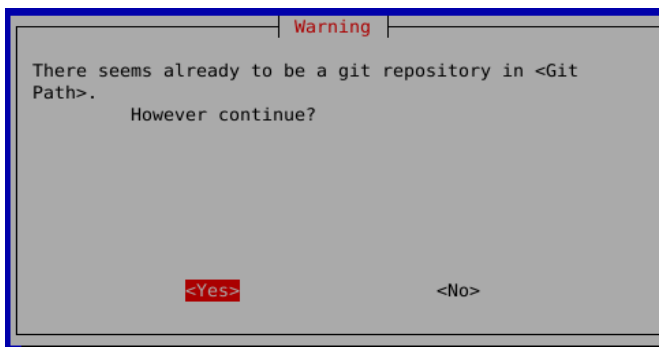


Abbildung 32.20.: Das Git-Repository existiert bereits.

32.4.1. Git-Repository anlegen

```

135b  <BuildNewPackage1 135b>≡ (135a)
      else
        echo "In ${GitPath} a new git repository will be created." >> ${log}
        git init
        <BuildNewPackage2 148a>

```

32.4.2. Name und E-Mail-Adresse ins Git-Repository einfügen

Der Name und die E-Mail-Adresse des Maintainers können in die Konfigurationsdatei des Git-Repositorys eingetragen werden, das für das Projekt angelegt worden ist. Diese Daten werden beispielsweise jeder Commit-Mitteilung hinzugefügt.

Zunächst wird abgefragt, ob dies erwünscht ist. Diese Information kann bereits höherrangig hinterlegt worden sein.

```
136a <BuildNewPackage3 136a>≡ (148a)
      if whiptail --title "Name and email" \
        --yesno "Do you like to add your name and email address \n \
        to the local git config file?" --yes-button "Yes" \
        --no-button "No" 15 60
      then
        AddNameAndEmail
      fi
<BuildNewPackage5 149a>
```

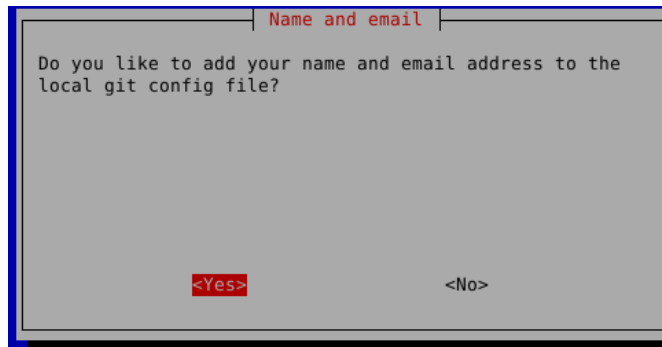


Abbildung 32.21.: Name und E-Mail.

Die folgende Funktion wird immer dann aufgerufen, wenn bei der Anlage eines neuen Git-Repositorys die Frage nach der Eintragung in die lokale Konfigurationsdatei bejaht wird.

```
136b <AddNameAndEmail 136b>≡ (147)
      function AddNameAndEmail {
        # Called by BuildNewPackage CloneFromSalsa
        # ImportDebianPackage and itself

        DEBValues
        GCName=${DEBFULLNAME}
        GCEmail=${DEBEMAIL}

        <AddNameAndEmail1 144b>
```

Damit der Paketierer nicht immer seinen vollen Namen und seine E-Mail-Adresse schreiben muss, sucht das Skript diese Daten zunächst in der Konfigurationsdatei und sodann in der `~/.bashrc`. Gegebenenfalls wird Gefundenes oder Erfragtes in die Konfigurationsdatei eingetragen.

```
136c <Bashrc 136c>≡
      DEBEMAIL="your.email.address@example.org"
      DEBFULLNAME="Firstname Lastname"
      export DEBEMAIL DEBFULLNAME
```

```

137  <DEBValues 137>≡ (138)
      function DEBValues {
          # Called by DebianControlTemplate PatchHeader AddNameAndEmail
          MaintainerF=0
          set +e
          # if Maintainer is stored in config file
          if [ "${Maintainer}" ]
          then
              Maintainer=$(echo ${Maintainer} | sed --expression='s/_/ /g')
              Maintainer=$(echo ${Maintainer} | sed --expression='s/@lt@/</g')
              Maintainer=$(echo ${Maintainer} | sed --expression='s/@gt@/>/g')
              DEBFULLNAME=$(echo ${Maintainer} | sed --expression='s/<.*//')
              DEBEMAIL=$(echo ${Maintainer} | sed --expression='s/^.*</' | sed 's/>/'')
              MaintainerF=1
          fi

          # if Uploaders is stored in config file
          if [ "${Uploaders}" ]
          then
              Uploaders=$(echo ${Uploaders} | sed --expression='s/_/ /g')
              Uploaders=$(echo ${Uploaders} | sed --expression='s/@lt@/</g')
              Uploaders=$(echo ${Uploaders} | sed --expression='s/@gt@/>/g')
              DEBFULLNAME=$(echo ${Uploaders} | sed --expression='s/<.*//')
              DEBEMAIL=$(echo ${Uploaders} | sed --expression='s/^.*</' | sed 's/>/'')
          fi

          # Looking for a team as maintainer
          if [ ${MaintainerF} -eq 0 ]
          then
              TeamMaintainer
          fi
      }
<DEBValues3 139a>

```

Es gibt Pakete, die von einem Team^[18] betreut werden. Meist werden viele gleichartige Pakete von einem solchen Team betreut. In diesen Fällen tritt das Debian-Projektmitglied als *Uploader* auf und das Team als *Maintainer*. Dies wird entsprechend in die Datei *debian/control* (Kapitel 35.4.6, Seite 235) eingetragen. Im Programmskript werden bisher das *Java packaging team*, das *Mozilla WebExtensions packaging team* und das *Debian Python Team* berücksichtigt.

```

138 <TeamMaintainer 138>≡ (235a)
    function TeamMaintainer {
        # Called by DEBValues

        # Makes sure that variable exists
        if [ -z '${JavaFlag}' ]
        then
            JavaFlag = 0
        fi

        if [ ${JavaFlag} -eq 1 ]
        then
            Maintainer="Debian Java Maintainers \
            <pkg-java-maintainers@lists.aliioth.debian.org>"
            MaintainerF=1
        fi

        if [ -z '${WebextFlag}' ]
        then
            WebextFlag = 0
        fi

        if [ ${WebextFlag} -eq 1 ]
        then
            Maintainer="Debian Mozilla Extension Maintainers \
            <pkg-mozext-maintainers@aliioth-lists.debian.org>"
            MaintainerF=1
        fi

        if [ -z '${PythonFlag}' ]
        then
            PythonFlag = 0
        fi

        if [ ${PythonFlag} -eq 1 ]
        then
            Maintainer="Debian Python Team <team+python@tracker.debian.org>"
            MaintainerF=1
        fi
    }

    <DEBValues 137>

```

```

139a  <DEBValues3 139a>≡ (137)
# Extracts DEBFULLNAME and DEBEMAIL from ~/.bashrc (if exist)
if [ ${MaintainerF} -eq 0 ]
then
  if grep --quiet 'DEBFULLNAME' ~/.bashrc
  then
    dfnb=$(grep DEBFULLNAME ~/.bashrc)
    dfnb=$(echo ${dfnb} | sed --expression='s/export .*//')
    dfnb=$(echo ${dfnb} | sed --expression='s/DEBFULLNAME=//')
    dfnb=$(echo ${dfnb} | sed --expression='s/"//g')
    dfnb=$(echo ${dfnb} | sed --expression="s/'//g")
    DEBFULLNAME=${dfnb}
  fi
  if grep --quiet 'DEBEMAIL' ~/.bashrc
  then
    demb=$(grep 'DEBEMAIL' ~/.bashrc)
    demb=$(echo ${demb} | sed --expression='s/export .*//')
    demb=$(echo ${demb} | sed --expression='s/DEBEMAIL=//')
    demb=$(echo ${demb} | sed --expression='s/"//g')
    demb=$(echo ${demb} | sed --expression="s/'//g")
    DEBEMAIL=${demb}
  fi
  Maintainer=${DEBFULLNAME}" <${DEBEMAIL}">"
  MaintainerF=1
fi

# Insert name and email address
if [ ${MaintainerF} -eq 0 ]
then
  DEBFULLNAME=$(whiptail --title "Name of the maintainer" \
  --inputbox "Please insert full name of the maintainer" \
  --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
<DEBValues4-1 139b>

```

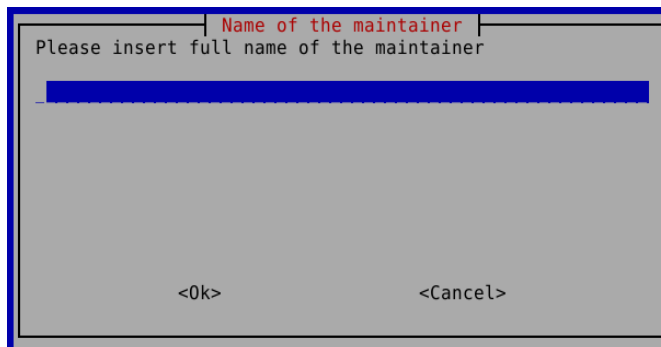


Abbildung 32.22.: Name des Maintainer

```

139b  <DEBValues4-1 139b>≡ (139a)
DEBEMAIL=$(whiptail --title "Email of the maintainer" \
--inputbox "Please insert email address of the maintainer" \
--cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
<DEBValues4-2 140a>

```

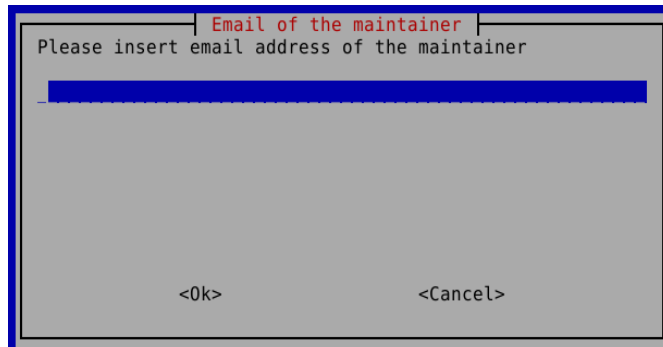


Abbildung 32.23.: E-Mail des Maintainers

```

140a <DEBValues4-2 140a>≡ (139b)
      Maintainer=${DEBFULLNAME}" <${DEBEMAIL}>"
      changeflag=1
      MaintainerF=1
      fi

      if ! whiptail --title "Maintainer" \
        --yesno "The full name and email address of the maintainer(s):\n \
          ${Maintainer}" --yes-button "OK" --no-button "Insert other" 15 60
        <DEBValues5 140b>

```

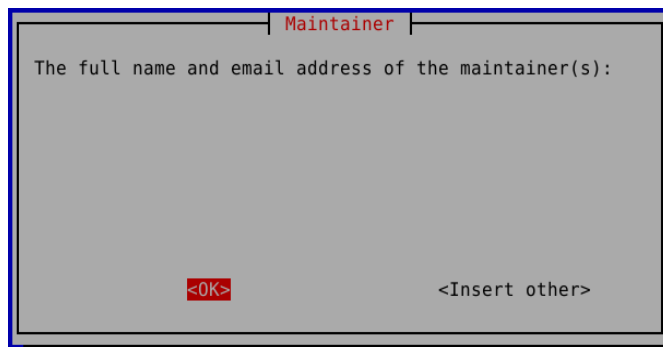


Abbildung 32.24.: Debian-Maintainer OK?

```

140b <DEBValues5 140b>≡ (140a)
      then
        DEBFULLNAME=$(whiptail --title "Name of the maintainer" \
          --inputbox "Please insert full name of the maintainer" \
          --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
        <DEBValues5-1 141a>

```

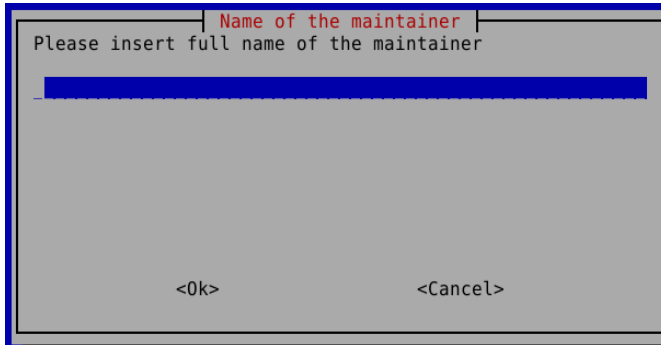



Abbildung 32.25.: Name des Debian-Maintainer

141a `<DEBValues5-1 141a>≡` (140b)
`DEBEMAIL=$(whiptail --title "Email of the maintainer"\
--inputbox "Please insert email address of the maintainer" \
--cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
<DEBValues5-2 141b>`

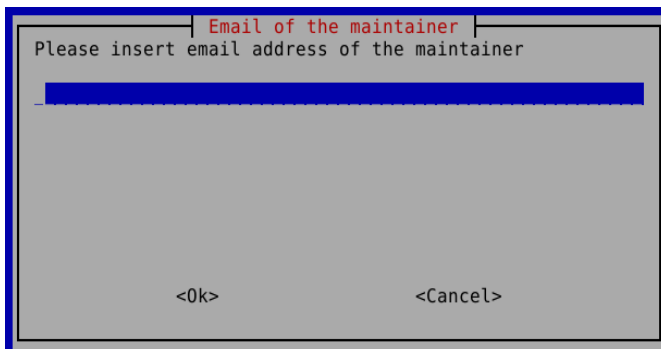


Abbildung 32.26.: E-Mail des Debian-Maintainer

141b `<DEBValues5-2 141b>≡` (141a)
`Maintainer=${DEBFULLNAME}" <${DEBEMAIL}>"
change flag=1
MaintainerF=1
fi
<DEBValues6 142a>`

142a $\langle DEBValues6\ 142a \rangle \equiv$ (141b)

```
# Insert maintainer data into config file if necessary
if [ $changeflag -eq 1 ]
then
  # Because Maintainer contains blanks
  MaintainerCF=$(echo ${Maintainer} | sed --expression='s/ /_/g')
  # Remove < and >
  MaintainerCF=$(echo ${MaintainerCF} | sed --expression='s/</@lt@g/')
  MaintainerCF=$(echo ${MaintainerCF} | sed --expression='s/>/@gt@g/')
  echo '## Maintainer and Uploaders' >> ${ConfigPath}${OrigName}
  echo 'Maintainer=${MaintainerCF}' >> ${ConfigPath}${OrigName}
  changeflag=0
fi
```

$\langle DEBValues7\ 142b \rangle$

142b $\langle DEBValues7\ 142b \rangle \equiv$ (142a)

```
# Insert uploaders data into config file if necessary
if [ ${JavaFlag} -eq 1 ]
then
  if [ -z "${Uploaders}" ]
  then
    if grep --quiet 'DEBFULLNAME' ~/.bashrc
    then
      dfnb=$(grep DEBFULLNAME ~/.bashrc)
      dfnb=$(echo ${dfnb} | sed --expression='s/export .*//')
      dfnb=$(echo ${dfnb} | sed --expression='s/DEBFULLNAME=//')
      dfnb=$(echo ${dfnb} | sed --expression='s"/"/g')
      dfnb=$(echo ${dfnb} | sed --expression="s/'/'g")
      DEBFULLNAME=${dfnb}
    else
      DEBEMAIL=$(whiptail --title "Email of the uploader" \
        --inputbox "Please insert email address of the uploader" \
        --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
    fi
  fi
```

$\langle DEBValues8\ 143 \rangle$

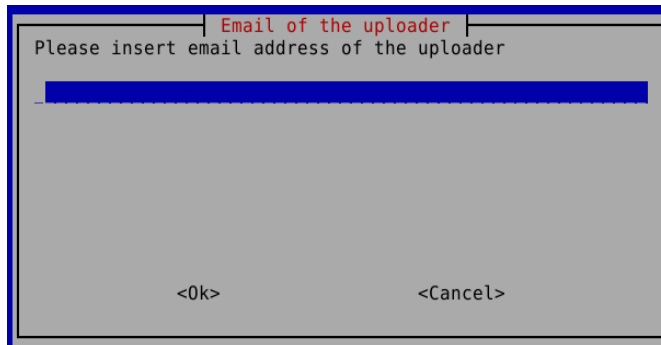


Abbildung 32.27.: E-Mail des Debian-Uploaders

```

143  <DEBValues8 143>≡ (142b)
      if grep --quiet 'DEBEMAIL' ~/.bashrc
      then
          demb=$(grep 'DEBEMAIL' ~/.bashrc)
          demb=$(echo ${demb} | sed --expression='s/export .*//')
          demb=$(echo ${demb} | sed --expression='s/DEBEMAIL=//')
          demb=$(echo ${demb} | sed --expression='s/"//g')
          demb=$(echo ${demb} | sed --expression="s/'//g")
          DEBEMAIL=${demb}
      else
          DEBEMAIL=$(whiptail --title "Email of the uploader" \
          --inputbox "Please insert email address of the uploader" \
          --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
      fi
<DEBValues9 144a>

```

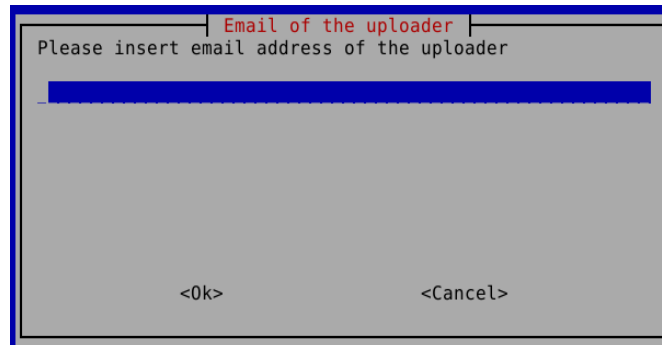


Abbildung 32.28.: E-Mail des Debian-Uploaders

```

144a  <DEBValues9 144a>≡ (143)
        Uploaders=${DEBFULLNAME}" <${DEBEMAIL}>"
        # Because Uploaders contains blanks
        UploadersCF=$(echo ${Uploaders} | sed --expression='s/ /_/g')
        # Remove < and >
        UploadersCF=$(echo ${UploadersCF} | sed --expression='s/</@lt@/g')
        UploadersCF=$(echo ${UploadersCF} | sed --expression='s/>/@gt@/g')
        echo 'Uploaders=${UploadersCF}' >> ${ConfigPath}${OrigName}
        changeflag=0
    fi
fi
set -e
}

```

<DebianControlTemplate 235b>

```

144b  <AddNameAndEmail1 144b>≡ (136b)
        if [ -z "${GCName}" ]
        then
            InsertDebName
        fi
        # if [ -n "${GCName}" ]
        # then
            git config user.name ${GCName}
        # fi
        if [ -z "${GCEmail}" ]
        then
            InsertDebEmail
        fi
        # if [ -n "${GCEmail}" ]
        # then
            git config user.email ${GCEmail}
        # fi

        set +e
        configStr=$(git config --list | grep 'user')
        set -e

        if ! whiptail --title "Result" \
            --yesno "${configStr}\nAllright?" \
            --yes-button "Yes" --no-button "No" 15 60
        <AddNameAndEmail2 145a>

```

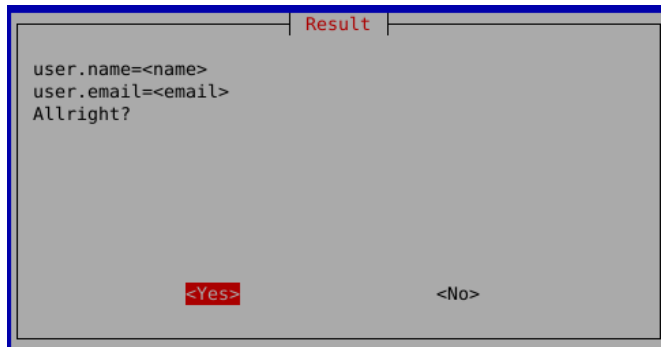


Abbildung 32.29.: Name und E-Mail des Maintainers korrekt?

```
145a  <AddNameAndEmail2 145a>≡ (144b)
      then
        AddNameAndEmail
      fi
}
```

<AddHomeServer 149b>

Die folgende Funktion dient zur Eingabe des Vor- und Nachnamens des Maintainers.

```
145b  <InsertDebName 145b>≡ (130b)
      function InsertDebName {

          # Called by AddNameAndEmail and itself
          DEBFULLNAME=$(whiptail --title "Name of the maintainer" \
            --inputbox "Please insert full name of the maintainer" \
            --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)
      }
      <InsertDebName1 146a>
```

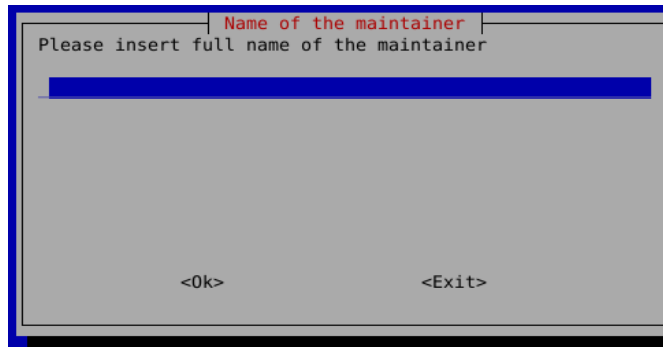


Abbildung 32.30.: Name des Maintainers eingeben

```

146a  <InsertDebName1 146a>≡                                     (145b)
      if [ $? -ne 0 ]
      then
        exit
      fi

      # Test Name
      if [ -z "${DEBFULLNAME}" ]
      then
        whiptail --title "Your Name" \
          --msgbox "Your name is necessary." 15 60
        InsertDebName
      fi
    }
    <InsertDebEmail 146b>

```

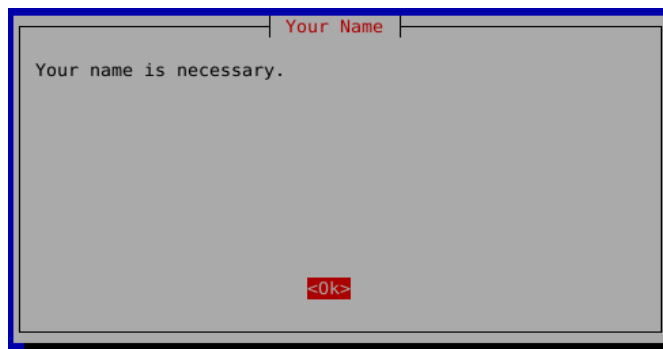


Abbildung 32.31.: Name des Maintainers erforderlich

Die folgende Funktion dient zur Eingabe des E-Mail-Adresse des Maintainers.

```

146b  <InsertDebEmail 146b>≡                                     (146a)
      function InsertDebEmail {

        # Called by AddNameAndEmail and itself
        DEBEMAIL=$(whiptail --title "Email of the maintainer" \
          --inputbox "Please insert email address of the maintainer" \
          --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)
      }
    <InsertDebEmail1 147>

```

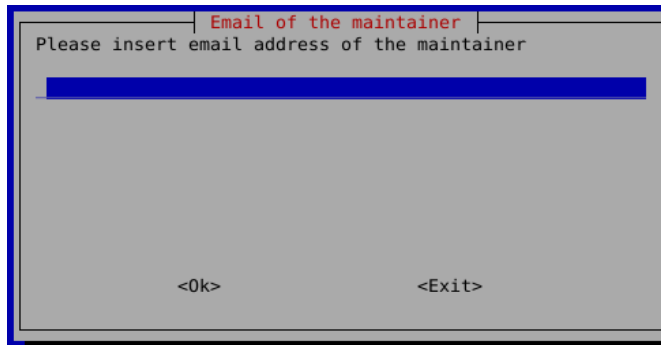


Abbildung 32.32.: E-Mail-Adresse des Maintainers eingeben

```

147  <InsertDebEmail 147>≡ (146b)
      if [ $? -ne 0 ]
      then
          exit
      fi

      # Regex string
      EmailR="\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,6}\b"

      # Test email address
      set +e
      if ! echo ${DEBEMAIL} | grep -E ${EmailR} > /dev/null
      then
          whiptail --title "Bad!" --msgbox "That's no email address." 15 60
          InsertDebEmail
      fi
      set -e
  }
  <AddNameAndEmail 136b>

```

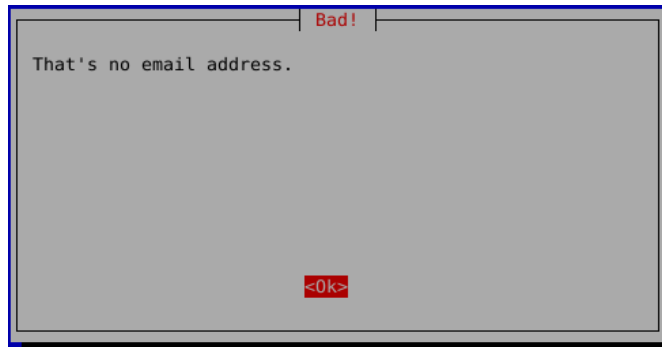


Abbildung 32.33.: Dies ist keine E-Mail-Adresse

32.4.3. Repositorium auf *salsa.debian.org*

Das Salsa-Repositorium wird als „Remote-Repository“ vom Programmskript eingetragen.

```
148a <BuildNewPackage2 148a>≡ (135b)
      git remote add salsa git@salsa.debian.org:${SalsaName}
```

```
<BuildNewPackage3 136a>
```

32.4.3.1. Manuell

Das entsprechende Repositorium auf *salsa.debian.org* wird dann manuell (Kapitel 21.2, Seite 75) angelegt (s. Kapitel 50.1, Seite 409).

Hieran erinnert das Programmskript auch den Nutzer.

```
148b <BuildNewPackage6 148b>≡ (149a)
```

```
      whiptail --title "If not happened yet:" \
      --msgbox "Please create a git repo ${SalsaName} \n \
      on salsa.debian.org!" 15 60
<BuildNewPackage6-1 150a>
```

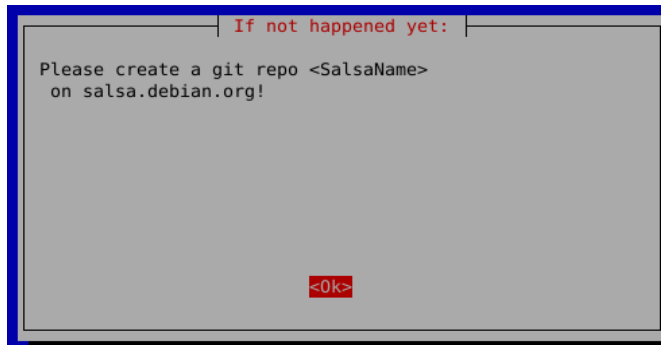



Abbildung 32.34.: Ein Git-Repository auf salsa anlegen

32.4.3.2. Innerhalb des Java-Teams

Innerhalb des Java-Teams (Kapitel 21.3, Seite 76) sollte ein neues Repository mit dem vom Team bereitgestellten *setup-salsa-repository*-Skript (Kapitel 50.1, Seite 409) angelegt werden.

32.4.4. Remoteserver anzeigen

Es werden die im Git-Repository eingetragenen Remoteserver angezeigt. Ein eigener Server wird nur angezeigt, wenn er zuvor eingetragen wurde.

```
149a <BuildNewPackage5 149a>≡ (136a)
      AddHomeServer
```

```
      fi
      <BuildNewPackage6 148b>
```

```
149b <AddHomeServer 149b>≡ (145a)
```

```
function AddHomeServer {
  # Called by BuildNewPackage AddGitServer ImportDebianPackage
  if [ -n "$ServerName" ]
  then
    git remote add home $(whoami)@${ServerName}:/srv/git/${SourceName}.git

    whiptail --title "Remoteserver" \
      --msgbox "New server added:\n$(git remote --verbose)" 15 60
    echo -e "New server added:\n $(git remote --verbose)" >> ${log}
  fi
}
```

```
<BuildNewPackage 135a>
```

```

Remoteserver
New server added:
home
mechtilde@pontian:/srv/git/buildscript-nw.git (fetch)
home
mechtilde@pontian:/srv/git/buildscript-nw.git (push)
salsa      git@salsa.debian.org:ddp-team/dpb.git
(fetch)
salsa      git@salsa.debian.org:ddp-team/dpb.git
(push)
<0k>

```

Abbildung 32.35.: Remoteserver ergänzen

Sodann wird eine neue Version heruntergeladen (Kapitel 34.2, Seite 183).

```

150a <BuildNewPackage6-1 150a>≡ (148b)
      NoPull=1 # Do not ask for pulling from salsa.debian.org
      Task=2 # Go to BuildNewVersion
    }

```

<IdentifyBranches 154a>

NoPull=1 verhindert, dass vor dem Herunterladen der neuen Version gefragt wird, ob zuvor eventuelle Änderungen von *salsa.debian.org* heruntergeladen werden sollen.

32.5. Klonen von *salsa.debian.org*

In diesem Fall entfallen teilweise Schritte, die in den anderen Kapiteln genannt sind. Das Klonen erfolgt mit dem Befehl *gbp clone*. Danach wird beim erfolgreichen Klonen das Remote-Repository von *origin* nach *salsa* umbenannt. Damit ist eine aussagekräftigere Benennung der Repositorien möglich.

Andernfalls erfolgt eine Fehlermeldung und das Programm endet.

```

150b <CloneFromSalsa 150b>≡ (182b)
      function CloneFromSalsa {
        # Called by TaskSelect
        echo "Clone an existing repo from salsa.debian.org" >> ${log}
        cd ${PrjPath}
        gbp clone git@salsa.debian.org:${SalsaName} --aliases ${SourceName}
        if [ $? -eq 0 ]
        then
          cd ${GitPath}
          git remote rename origin salsa
          echo "${SalsaName} was cloned" >> ${log}
        else
          echo "${SalsaName} could not be cloned"
          exit
        fi
      }

```

<CloneFromSalsa2 151a>

Bei einem erfolgreichen Herunterladen werden mittels der Funktion *DebianBranchName* die heruntergeladenen Zweige (Branches) angezeigt und der aktive Zweig gesondert angezeigt.

```
151a <CloneFromSalsa2 151a>≡ (150b)
      # Identify branches and choose one
      DebianBranchName

<CloneFromSalsa3 155b>
```

32.5.1. Bestimmung der Git-Zweige

Mit der Funktion *DebianBranchName* werden die Git-Zweige des geklonten Salsa-Repositoriums ermittelt.

Hierzu wird zunächst die Funktion *IdentifyBranches* (Kapitel 32.5.2, Seite 154) ausgeführt.

```
151b <DebianBranchName 151b>≡ (155a)
      function DebianBranchName {
          # Called by CloneFromSalsa

          ## Identify and show branches
          IdentifyBranches
          ba=($b1)

          whiptail --title "Branches in repo ${OrigName}:" --msgbox "${b1}" 15 60

<DebianBranchName2 151c>
```

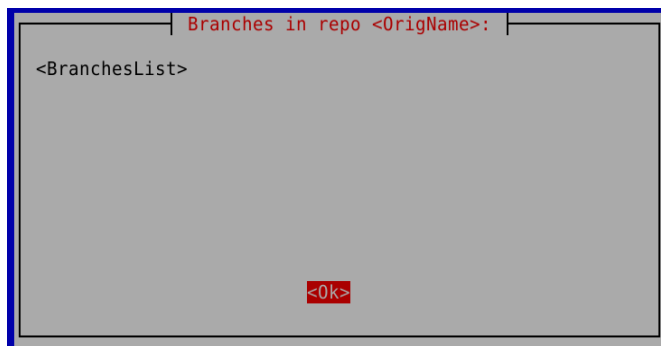


Abbildung 32.36.: Zeigt Liste der Git-Zweige

Sodann wird der aktive Git-Zweig ermittelt.

```
151c <DebianBranchName2 151c>≡ (151b)
      set +e
      for element in ${ba[*]}
      do
          # Find the default branch
          if echo ${element} | grep --quiet '^x_'
          then
              DefaultBranch=$(echo ${element} | sed --expression='s/^x_//')
              whiptail --title "Recent branch found" \
                  --msgbox "Found: ${DefaultBranch}" 15 60
          fi
<DebianBranchName3 152a>
```

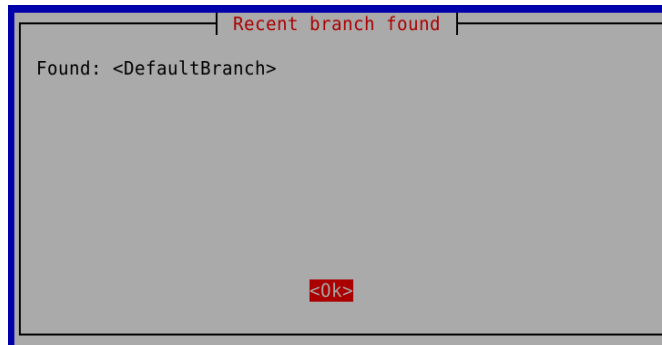


Abbildung 32.37.: Zeigt aktuellen Git-Zweig

```

152a  <DebianBranchName3 152a>≡ (151c)
      # Ignore HEAD
      if echo ${element} | grep --quiet 'HEAD'
      then continue
      fi
      # Checkout all branches
      if echo ${element} | grep --quiet '^remotes/salsa/'
      then
          NewBranchName=$(echo ${element} | \
              sed --expression='s/^remotes\/salsa\/\\\\/')
          git checkout ${NewBranchName}
          whiptail --title "Checkout branch" \
              --msgbox "Checkout of ${NewBranchName}" 15 60
      fi
      done
      set -e
  <DebianBranchName4 152b>

```

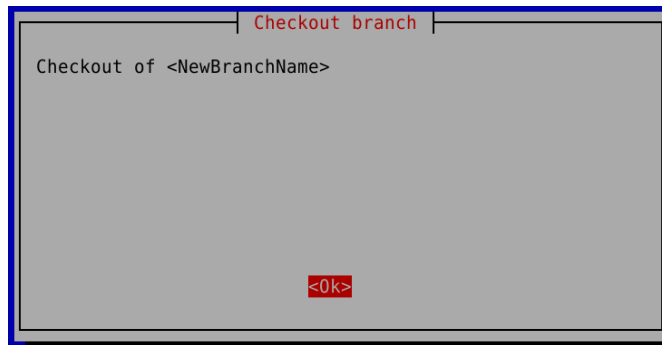


Abbildung 32.38.: Aktueller Git-Zweig

```

152b  <DebianBranchName4 152b>≡ (152a)

      # Finally checkout the default branch (again)
      git checkout ${DefaultBranch}
      whiptail --title "Checkout branch" \
          --msgbox "Checkout of ${DefaultBranch}" 15 60
  <DebianBranchName5 153a>

```

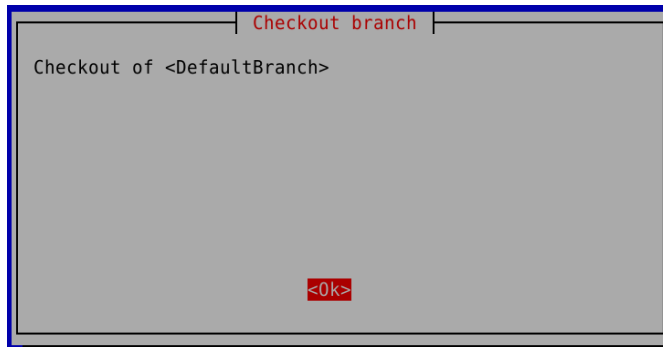


Abbildung 32.39.: Vorgegebener Git-Zweig

153a `<DebianBranchName5 153a>≡`

(152b)

```
# Insert default branch into the config file and write into logfile
echo 'DefaultBranch=${DefaultBranch} >> ${ConfigPath}${OrigName}'
echo "Branches: "${bl} >> ${log}
echo "DefaultBranch: "${DefaultBranch} >> ${log}
RecentBranch=${DefaultBranch}
whiptail --title "Please check! (1)" \
--msgbox "The branch is ${RecentBranch}" 15 60
<DebianBranchName6 153b>
```

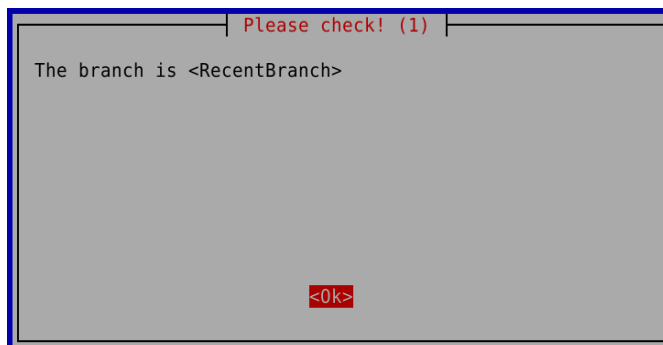


Abbildung 32.40.: Aktueller Git-Zweig

153b `<DebianBranchName6 153b>≡`

(153a)

```
echo 'RecentBranch=${RecentBranch} >> ${ConfigPath}${OrigName}'
echo "RecentBranch: "${RecentBranch} >> ${log}
Distro4Branch
}

<FailureNotice 171>
```

Durch den Aufruf der Funktion *Distro4Branch* wird schließlich dem Git-Zweig eine Distribution zugeordnet (Kapitel 32.5.3, Seite 154)

32.5.2. Git-Zweige ermitteln

Diese Funktion ermittelt alle vorhandenen Zweige. Dies geschieht mittels des Befehles *git branch --all*. Bei der Kennzeichnung des aktiven Zweiges werden Sternchen (*) und Leerzeichen durch kleines X (x) und Unterstrich(_) ersetzt.

Diese Funktion wird an verschiedenen Stellen im Ablauf des Programmskriptes aufgerufen.

```
154a <IdentifyBranches 154a>≡ (150a)
function IdentifyBranches {
    # Called by DebianBranchName AskDist DebianBranches
    cd ${GitPath}
    # sed is used to kill the asterisk
    bl=$(git branch --all | sed --expression='s/* /x_/')
}

<CreateNewCow 313>
```

32.5.3. Git-Zweig Distribution zuordnen

An verschiedenen Stellen im Programmablauf ist es erforderlich, einem Git-Zweig eine Debian-Veröffentlichung zuzuordnen.

```
154b <Distro4Branch 154b>≡ (314)
function Distro4Branch {
    # Called by DebianBranchName CreateNewBranch AskDist BuildNewRevision

    set +e
    # Set Debian distribution for branch
    if [ ${RecentBranch} != "debian/sid" -o -z "${RecentBranchD}" ]
    then
        CowL=$(ls /var/cache/pbuilder/ | grep .cow | grep '-' | \
            sed 's/base-//' | sed 's/.cow//')
        CowA=(${CowL})

        i=1; cowE="0 sid on "
        for element in ${CowA[*]}
        do
            cowE=${cowE} '$i' '${element}' off '
            i=$(expr $i + 1)
        done

        RecentBranchDnr=$(whiptail --title "Debian release" \
            --radiolist "Select pbuilder cow:" \
            --cancel-button "Other" 15 60 8 \
            $cowE 3>&2 2>&1 1>&3)
    <Distro4Branch2 155a>
```

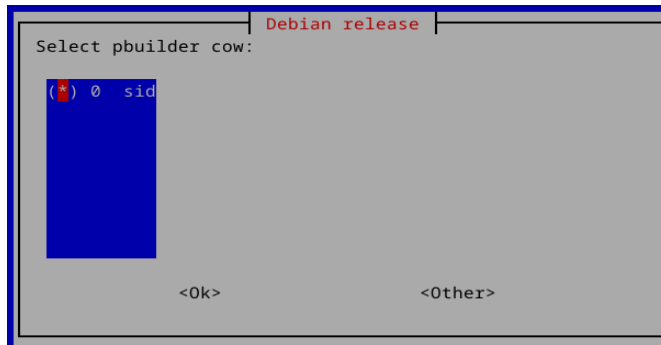


Abbildung 32.41.: Auswahl des Debian Release.

```

155a  <Distro4Branch2 155a>≡ (154b)
      if [ $? -eq 1 ]
      then
          CreateNewCow
      fi

      if [ ${RecentBranchDNr} -eq 0 ]
      then
          bDist="sid"
      else
          RecentBranchDNr=$(expr ${RecentBranchDNr} - 1)
          bDist=${CowA[${RecentBranchDNr}]}
      fi

      echo "# ${RecentBranch}"_Dist="${bDist} >> ${ConfigPath}${OrigName}
      RecentBranchD=${bDist}
      echo "Notice from Distro4Branch: The distribution is " \
      ${RecentBranchD} >> ${log}

      fi
      set -e
  }

  <DebianBranchName 151b>

```

32.5.4. Name und E-Mail-Adresse hinzufügen

Besonders für den Fall, dass es sich um kein „eigenes“ Salsa-Repository handelt, wird die Möglichkeit eröffnet, den Namen und die E-Mail-Adresse des Maintainers in die lokale git-Konfigurationsdatei einzutragen. Dies erfolgt mit der Funktion *AddNameAndEmail* (s. a, Kapitel 32.4.2, Seite 136

```

155b  <CloneFromSalsa3 155b>≡ (151a)
      if whiptail --title "Name and email" \
      --yesno "Do you like to add your name and email address \n \
      to the local git config file?" --yes-button "Yes" \
      --no-button "No" 15 60
      <CloneFromSalsa5 156a>

```

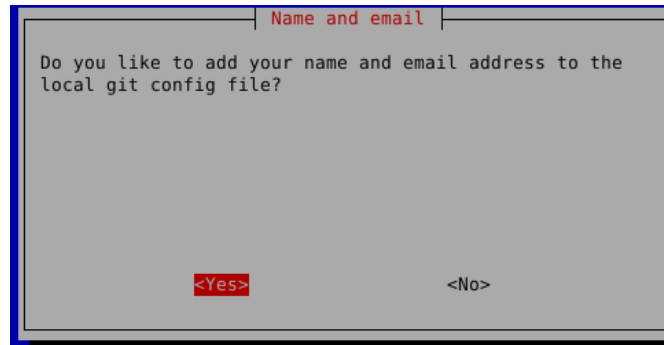


Abbildung 32.42.: Name und E-Mail.

```

156a  <CloneFromSalsa5 156a>≡ (155b)
      then
      AddNameAndEmail
      fi

      <CloneFromSalsa7 156b>

156b  <CloneFromSalsa7 156b>≡ (156a)
      AddHomeServer
      CommonTasks
    }

      <CutSuffix 192b>

```

32.6. Import eines Debian-Quellcode-Paketes

Nicht alle Pakete des Debian-Projektes liegen auf *salsa.debian.org*, einer Gitlab-Instanz. Von den Paketen auf *salsa.debian.org* werden nicht alle mit *git-buildpackage* gebaut. Manchmal fehlt der Zweig *pristine-tar*. In diesem Fall ist ein Klonen nicht sinnvoll.

Auch diese Pakete können jedoch mithilfe dieses Programmskriptes gebaut werden. In der Datei */etc/apt/sources.list* oder in einer Datei im Verzeichnis */etc/apt/sources.list.d* muss dazu der Eintrag

```
deb-src http://deb.debian.org/debian/ sid main
```

aktiviert vorhanden sein. Dies bedeutet, dass nach der Aktivierung auch *sudo apt update* ausgeführt werden muss.

Für seltene Sonderfälle ist diese Zeile entsprechend anzupassen.

Dieser Eintrag ermöglicht das Herunterladen von Quellpaketen.

```

156c  <ImportDebianPackage 156c>≡ (339b)
      function ImportDebianPackage {
          # Called by TaskSelect

          echo "Import an existing package without pristine-tar" >> ${log}
          cd ${PrjPath}
          echo "Download ${SourceName} with apt source" >> ${log}
          apt source --download-only ${SourceName} >> ${log}

          <ImportDebianPackage2 157a>

```


Mit `apt source --download-only` wird nur das Quellcodepaket heruntergeladen.

Das Erstellen des Git-Verzeichnisses erfolgt mit `gbp import-dsc` und der heruntergeladenen `*.dsc`-Datei.

Da die Funktion `gbp import-dsc` ein Signieren ausführt, wird durch die Funktion `GpgKeyAvailable` (Kapitel 32.8, Seite 160) zunächst abgefragt, ob der GnuPG-Schlüssel verfügbar ist.

```
157a <ImportDebianPackage2 157a>≡ (156c)
      GpgKeyAvailable
```

```
# GitPath has to be deleted because gbp import-dsc will create it.
rmdir --verbose ${GitPath} >> ${log}
```

```
echo "gbp import-dsc" >> ${log}
gbp import-dsc --verbose ${SourceName}*.dsc >> ${log}
```

```
cd ${GitPath}
git remote add salsa git@salsa.debian.org:${SalsaName}
<ImportDebianPackage3 157b>
```

Danach erfolgt die Anzeige der vorhandenen Git-Zweige und die Bestimmung des aktiven Zweiges durch Aufruf der Funktion `DebianBranchName` (Kapitel 32.5.1, Seite 151).

Dann wird die Möglichkeit eröffnet, den Namen und die E-Mail-Adresse des Maintainers in die lokale git-Konfigurationsdatei einzutragen. Dies erfolgt mit der Funktion `AddNameAndEmail` (s, a, Kapitel 32.4.2, Seite 136

```
157b <ImportDebianPackage3 157b>≡ (157a)
```

```
# Identify branches and choose one
DebianBranchName
```

```
if whiptail --title "Name and email" \
--yesno "Do you like to add your name and email address \n \
to the local git config file?" --yes-button "Yes" \
--no-button "No" 15 60
```

```
<ImportDebianPackage3a 157c>
```

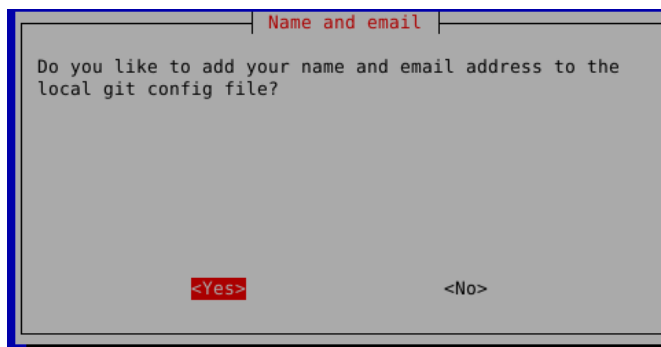


Abbildung 32.43.: Name und E-Mail.

```
157c <ImportDebianPackage3a 157c>≡ (157b)
```

```
then
  AddNameAndEmail
fi
```

```
<ImportDebianPackage4 158a>
```

4. Mai 2025

```
158a <ImportDebianPackage4 158a>≡ (157c)
    AddHomeServer
    PQImport
    CommonTasks

}

<Import4Sponsoring 158b>
```

32.7. Import für das Sponsoring

Ein spezieller Fall für die Erstellung eines Git-Repositoriums mit *gpb import-dsc* ist das Sponsoring. Das Herunterladen der Quellcodepakete erfolgt hier mit *dget* (in der Regel von *mentors.debian.net*).

```
158b <Import4Sponsoring 158b>≡ (158a)
    function Import4Sponsoring {
        # Called by StartTasks

        cd ${PrjPath}

        if whiptail --title "Should the Debian sources be downloaded?" \
            --yesno "Should the Debian sources (*.orig.tar.gz, *.debian.tar.gz\n \
            and *.dsc) be downloaded?" \
            --defaultno --yes-button "Yes" --no-button "No" 15 60
        <Import4Sponsoring1 158c>
```

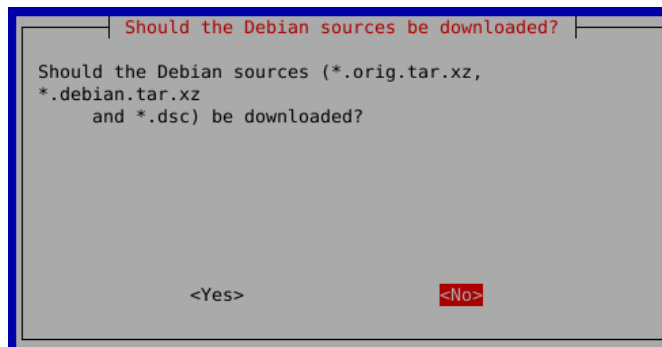


Abbildung 32.44.: Herunterladen der Debian Sourcen

Sollen die Quellcodepakete von *mentors.debian.net* heruntergeladen werden, hat die URL folgendes Format:

```
https://mentors.debian.net/debian/pool/main/<p> \
/<package name>/<package name>_x.y.z.dsc
```

```
158c <Import4Sponsoring1 158c>≡ (158b)
    then
        DownloadUrl=$(whiptail --title "Insert URL for download" \
            --inputbox "Please insert the complete\n \
            URL to download ${UpstreamSourceName}\n(with 'https://'\n \
            e.g. from mentors.debian.net):" \
            --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)

        <Import4Sponsoring2 159>
```

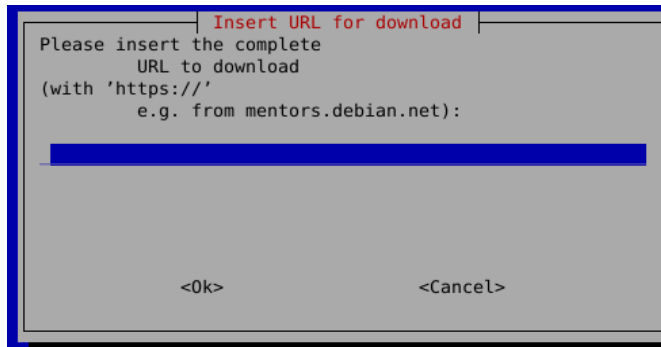


Abbildung 32.45.: Eingabe der Url der Debian Sourcen

```

159  <Import4Sponsoring2 159>≡ (158c)
      dget --download-only ${DownloadUrl}
      ls -la
      echo -e "\n Press RETURN to continue!"
      read a
    fi

    # GitPath has to be deleted because gbp import-dsc will create it.
    echo $(pwd) >> ${log}
    rmdir ${GitPath}

    GpgKeyAvailable
    gbp import-dsc --verbose $(ls *.dsc) >> ${log} &&

    ParseConfig
    cd ${SourceName}
    echo $(pwd) >> ${log}
    SBuildOrPBuilder 0
    Task=5 # Go to RunningTests
  }

  <CommonTasks 177>

```

Danach geht es mit dem Bauen des Debian-Paketes weiter (Kapitel 37.3.5, Seite 307).
Nach dem Bauen des Paket werden die Tests aufgerufen (Kapitel 40, Seite 327).

32.8. GnuPG-Schlüssel verfügbar?

Die Funktion *GpgKeyAvailable* fordert den Nutzer auf zu prüfen, ob der GnuPG-Schlüssel zum Signieren verfügbar ist.

Sie wird an verschiedenen Stellen im Programmskript aufgerufen. Wird die Frage nach der Verfügbarkeit des GnuPG-Schlüssels verneint, wird das Programmskript beendet.

```
160 <GpgKeyAvailable 160>≡ (168a)
    function GpgKeyAvailable {
        # Called by BuildWithUscan Import2Git CreateSignature ImportDebianPackage
        # PrepareUploading

        if ! whiptail --title "GPG-Key available?" \
            --yesno "GPG-Key must be available for signing." \
            --yes-button "Yes, is available" --no-button "Exit" 15 60
        then
            exit
        fi
        GettingFingerprint
    }

    <DetectPlugins 122>
```

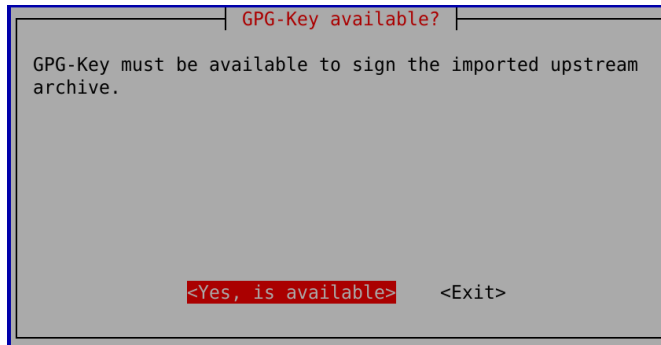


Abbildung 32.46.: GPG-Schlüssel verfügbar

32.9. Fingerprint nutzen

Zum Signieren wird der Fingerprint des Maintainer-Schlüssels benötigt. Dies ist der Fingerprint des Schlüssels, der auch im `Debian-Keyring` hinterlegt ist.

Befindet sich keine entsprechende Datei im Verzeichnis der Konfigurationsdateien, wird eine entsprechende Datei erstellt.

```
161 <GettingFingerprint 161>≡
    function GettingFingerprint {
        # Called by GpgKeyAvailable

        finchflag=0

        # getting the fingerprint of the key to sign
        if [ -f ${ConfigPath}/fingerprint ]
        then
            . ${ConfigPath}/fingerprint
        else
            mkdir --parents ${ConfigPath}
            finchflag=1
        fi

        if [ -z "${fipr}" ]
        then
            fipr=$(whiptail --title "Your fingerprint" \
                --inputbox "Please insert fingerprint of your key for signing!" \
                --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
        <GettingFingerprint1 162a>
```

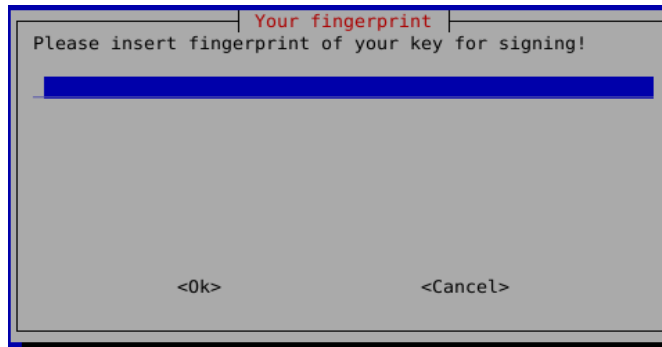


Abbildung 32.47.: Fingerprint eingeben

```
162a  <GettingFingerprint1 162a>≡ (161)
      if [ -z "${fipr}" ]
      then
          echo "Please insert fingerprint of your key for signing!"
          read fipr
      fi
      finchflag=1
  fi

  if ! whiptail --title "Fingerprint" \
  --yesno "Is ${fipr} the right fingerprint of the key for signing?" \
  --yes-button "Yes" --no-button "No" 15 60
  <GettingFingerprint2 162b>
```

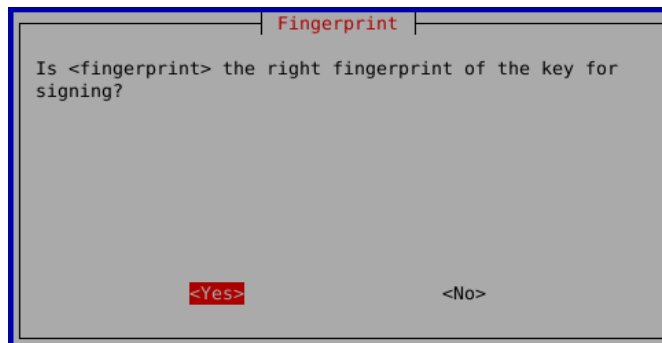


Abbildung 32.48.: Ist der Fingerprint korrekt?

```
162b  <GettingFingerprint2 162b>≡ (162a)
      then
          fipr=$(whiptail --title "Key for signing" \
          --inputbox "Real fingerprint of the key for signing:" \
          --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
  <GettingFingerprint3 163>
```

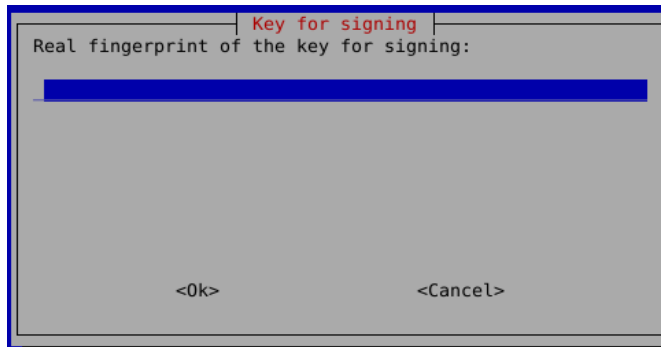


Abbildung 32.49.: Korrekten Fingerprint eingeben

```

163  <GettingFingerprint3 163>≡ (162b)
      if [ -z "${fipr}" ]
      then
          echo "Please insert fingerprint of your key for signing!"
          read fipr
      fi
      finchflag=1
  fi

  if [ $finchflag -eq 1 ]
  then
      if [ -f ${ConfigPath}/fingerprint ]
      then
          mv ${ConfigPath}/fingerprint ${ConfigPath}/fingerprint.backup
          whiptail --title "Fingerprint file" \
            --msgbox "You can find the old fingerprint file as\n \
            ${ConfigPath}/fingerprint.backup" 15 60
      fi
  fi
  <GettingFingerprint4 164a>

```

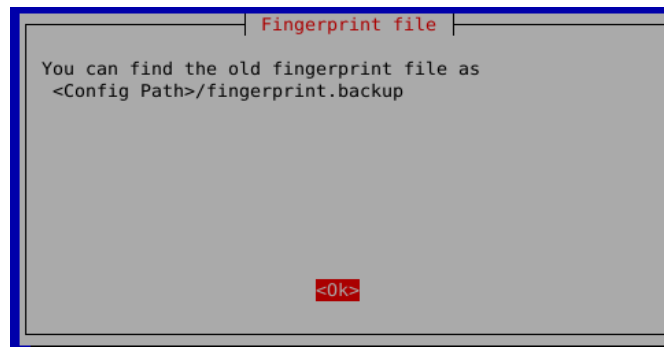


Abbildung 32.50.: Backup des Fingerprints anlegen

```
164a <GettingFingerprint4 164a>≡ (163)
    touch ${ConfigPath}/fingerprint
    echo "#!/usr/bin/bash" >> ${ConfigPath}/fingerprint
    echo "fipr=${fipr} >> ${ConfigPath}/fingerprint
    . ${ConfigPath}/fingerprint
    fi
}

<CreateSignature (nicht definiert)>
```

32.10. Start des Paketierens

Die Funktion *BuildApp* ruft am Ende die Aufgabenauswahl für die einzelnen Schritte des Paketierens auf.

```
164b <BuildApp10 164b>≡ (131b)
    set -e
    # Start of Packaging
    CommonTasks
}

<MainProgram 106a>
```


33. Arbeiten in einem angelegten Projekt

33.1. Konfigurationsdatei laden und editieren

Wird ein Projektname eingegeben, versucht das Programmskript die Konfigurationsdatei dieses Projektes zu laden und mit *less* anzuzeigen.

Die Funktion *ConfigFileLEC* zeigt die geladene Konfigurationsdatei an und fragt dann ab, ob sie korrekt sei. Dann kann entschieden werden, ob diese Datei editiert werden soll.

```
165a <ConfigFileLEC1 165a>≡ (111a)
      set +e
      if [ -f ${ConfigPath}${OrigName} ]
      then
        . ${ConfigPath}${OrigName} # executes config script
        less --LINE-NUMBERS ${ConfigPath}${OrigName}
        if ! whiptail --title "Check config file" \
          --yesno "Is the config file OK?" \
          --yes-button "Yes" --no-button "No" 15 60
        <ConfigFileLEC2 165b>
```

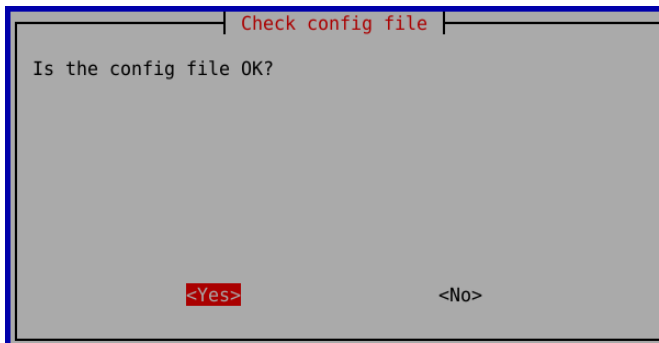


Abbildung 33.1.: Abfrage - Konfigurationsdatei.

Wird die Frage, ob die Konfigurationsdatei korrekt ist, verneint, kann sie editiert werden. Dies kann mit dem Editor *nano* geschehen oder durch die Beantwortung von Fragen.

Ansonsten geht es weiter mit Kapitel 33.4, Seite 169. Wenn jedoch nur ein oder kein Branch existiert, geht es mit Kapitel 33.4.7, Seite 176 weiter.

```
165b <ConfigFileLEC2 165b>≡ (165a)
      then
        Edit=$(whiptail --title "Edit Config File" \
          --radiolist "How do you like to edit the config file?" \
          15 60 2 "0" "Using Nano" on \
          "1" "Answering questions" off \
          --cancel-button "Exit" 3>&2 2>&1 1>&3)
        <ConfigFileLEC3 166a>
```

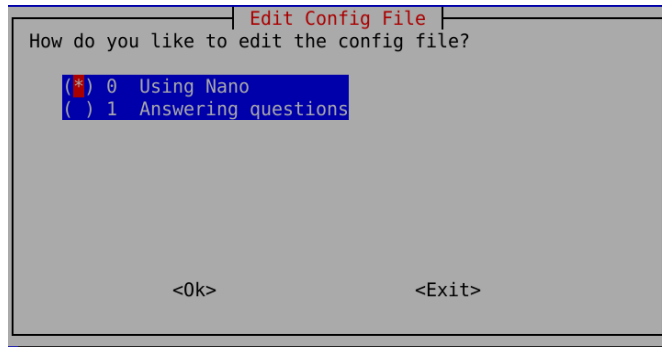


Abbildung 33.2.: Abfrage - Konfigurationsdatei bearbeiten.

Wird keine Editier-Methode angegeben, beendet sich das Programm.

166a $\langle \text{ConfigFileLEc3 } 166a \rangle \equiv$ (165b)

```

if [ -z "${Edit}" ]
then
    whiptail --title "Bye" --msgbox "Bye" 15 60
    exit
fi
 $\langle \text{ConfigFileLEc3-1 } 166b \rangle$ 

```

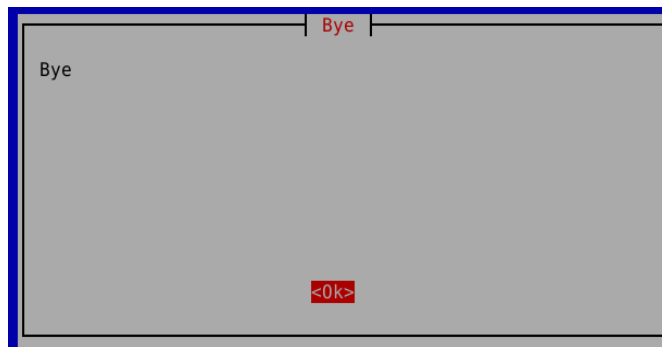


Abbildung 33.3.: Programm beendet

166b $\langle \text{ConfigFileLEc3-1 } 166b \rangle \equiv$ (166a)

```

if [ ${Edit} -eq 0 ]
then
    nano --linenumbers --mouse \
        --softwrap ${ConfigPath}${OrigName}
    . ${ConfigPath}${OrigName}
else
    AskConfig
fi
fi
. ${ConfigPath}${OrigName} # executes config script (another time)
SetDefaults
 $\langle \text{ConfigFileLEc4 } 111b \rangle$ 

```

Soll die Konfigurationsdatei mit dem Editor *nano* bearbeitet werden, wird dieser aufgerufen und öffnet sich. Nach der Beendigung desselben (mit *STRG-X*) läuft das Programmskript weiter (Kapitel 33.4, Seite 169).

Soll die Anpassung mittels Abfragen erfolgen, wird die Funktion *AskConfig* (s. Kapitel 32.1.1, Seite 113) aufgerufen, Danach wird die Datei neu geschrieben.

Enthält die Konfigurationsdatei hinsichtlich der Plugins keine Daten, so werden mit der folgenden Funktion Standardwerte gesetzt.

```
167a <SetDefaults 167a>≡ (128b)
function SetDefaults {
    # Called by ConfigfileLEC

    if [ -z "${WebextFlag}" ]
    then
        WebextFlag=0
    fi

    if [ -z "${PythonFlag}" ]
    then
        PythonFlag=0
    fi

    if [ -z "${JavaFlag}" ]
    then
        JavaFlag=0
    fi

    if [ -z "${MavenPluginFlag}" ]
    then
        MavenPluginFlag=0
    fi
}

<ConfigFileLEC 111a>
```

33.2. Ändern von Zeilen in der Konfigurationsdatei

Im Skript ist auch eine Funktion enthalten, die es ermöglicht, einzelne Werte in der Konfigurationsdatei zu ändern. Hierzu werden der Funktion zwei Parameter übergeben, nämlich als Erstes der Bezeichner der Variablen und zweitens der neue Wert derselben.

Diese Funktion wird von den Funktionen *BuildNewVersion* (Kapitel 34.3.2, Seite 186), *PQMigration* (Kapitel 36.2, Seite 257), *ChangeEntry* (Kapitel 33.4.5, Seite 174) und *OwnServer* (Kapitel 44.2, Seite 376) genutzt.

```
167b <ReplaceConfigLines 167b>≡ (168b)
function ReplaceConfigLines {
    # Called by BuildNewVersion PQMigration ChangeEntry OwnServer CurSuffix

    # This function needs two parameters:
    # First the name of the variable
    # and second it's new value
    ConfProp=$1
    ConfVal=$2

    <ReplaceConfigLines1 168a>
```

4. Mai 2025

Die Funktion prüft, ob die zu ändernde Zeile in der Konfigurationsdatei vorhanden ist. Andernfalls wird sie mit der Funktion *InsertConfigLines* erstellt.

Vor dem Ersetzen mit *sed* sind (ebenfalls mit *sed*) eventuell vorhandene Schrägstriche in der Variablen *ConfVal* zu maskieren.

```
168a <ReplaceConfigLines1 168a>≡ (167b)
    set +e
    cprop=$(grep --count ${ConfProp} ${ConfigPath}${OrigName})
    set -e
    if [ ${cprop} -ge 1 ]
    then
        # Masquerade slashes
        ConfVal=$(echo ${ConfVal} | sed 's/\\/\\/\\\\/g')

        sed --in-place --expression="s/${ConfProp}=.*/${ConfProp}=${ConfVal}/g" \
            ${ConfigPath}${OrigName}
    else
        # InsertConfigLine needs two parameters:
        # name of the variable and new value
        InsertConfigLine ${ConfProp} ${ConfVal}
    fi
}

<GpgKeyAvailable 160>
```

33.3. Zeile in Konfigurationsdatei einfügen

Die Funktion *InsertConfigLine* fügt eine Zeile in die Konfigurationsdatei ein. Ihr sind die gleichen Parameter wie der Funktion *ReplaceConfigLines* zu übergeben.

```
168b <InsertConfigLine 168b>≡
function InsertConfigLine {
    # Called by ReplaceConfigLines

    # This function needs two parameters:
    # First the name of the variable
    # and second it's value

    ConfProp=$1
    ConfVal=$2

    echo ${ConfProp}="${ConfVal}" >> ${ConfigPath}${OrigName}
}

<ReplaceConfigLines 167b>
```

33.4. Auswahl eines Git-Zweiges

Damit ein Wechsel des Git-Zweiges möglich ist, wird geprüft, ob im aktuellen Zweig unversionierte Änderungen vorhanden sind. Diese Prüfung erfolgt mit der Funktion *CheckGitStatus* (Kapitel 33.4.1, Seite 170)

Wenn kein Wechsel des Git-Zweiges erforderlich ist, müssen Änderungen im Verzeichnis *debian/* nicht zwingend versioniert werden.

Bereits erfolgte Änderungen am Upstream-Code sind mittels *git restore* zurückzusetzen.

Existieren mehrere Branches, kann ein Branch ausgewählt werden. Existiert nur ein Branch, so wird dieser genannt. Anderfalls wird die Funktion *StartTasks* aufgerufen (Kapitel 32.3, Seite 131).

Diese Funktion kann von der Aufgabenauswahl (Kapitel 33.5, Seite 177) aus aufgerufen werden.

Die Funktion *SelectBranch* wird gegebenenfalls auch beim Bauen einer neuen Revision aufgerufen (Kapitel 37.3.1, Seite 302).

```
169 <SelectBranch 169>≡ (175b)
    function SelectBranch {
        # Called by BuildApp TaskSelect BuildNewRevision

        CheckGitStatus
        DebianBranches
    }
    <SelectBranch1 173>
```

Nach der Funktion *CheckGitStatus*, die im folgenden Abschnitt beschrieben wird, wird die Funktion *DebianBranches* zur Ermittlung der lokalen *Debian*-Branches im *Git*-Repositorium aufgerufen (Kapitel 33.4.3, Seite 172).

33.4.1. Prüfung mit *git status*

Die Funktion *CheckGitStatus*, die auch noch an anderen Stelle im Programm-Skriptes aufgerufen wird, prüft, ob im aktuellen *Git*-Zweig neue oder geänderte Dateien vorhanden sind. Ist dies der Fall, können nämlich manche *Git*-Operationen nicht vorgenommen werden.

Wird dies mittels *git status* festgestellt, wird die Funktion *FailureNotice* aufgerufen und eine Möglichkeit zur Fehlerbehebung in einem weiteren Terminal eröffnet (Kapitel 33.4.2, Seite 171).

Gibt es keine Probleme, geht es weiter mit der Anzeige der vorhandenen *Git*-Zweige (Kapitel 33.4.4, Seite 173).

```

170 <CheckGitStatus 170>≡ (209b)
    function CheckGitStatus {
        # Called by Import2Git BuildWithUscan PQImport
        # PatchesTreatment SelectBranch and itself

        set +e
        # Checks git status
        echo "Notice from CheckGitStatus:" >> ${log}
        echo "$(git status) >> ${log}

        if [ ! -z "$(git status --short)" ]
        then
            git status
            FailureNotice "'git status' shows problems\n\
            Please clean up 'git status'"
            if whiptail --title "Check another time?" \
            --yesno "Do you want to check the git status another time?" \
            --yes-button "Yes" --no-button "No" 15 60
            then
                CheckGitStatus
            fi
        fi
        set -e
    }

    <CheckTags 216b>

```

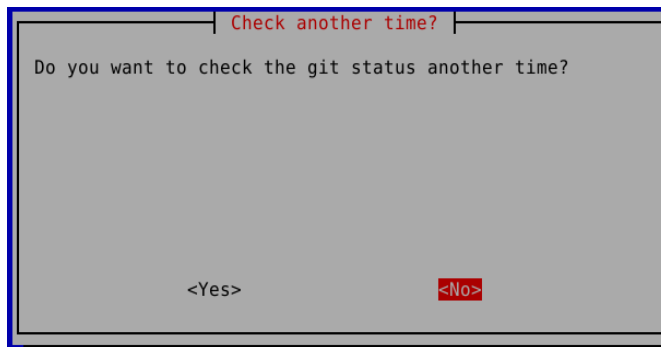


Abbildung 33.4.: Abfrage - Weiterer Check?.

33.4.2. Fehlermeldung und -behebung

Wird die Funktion *FailureNotice* aufgerufen, erfolgt eine Meldung auf dem Terminal und es wird die Möglichkeit eröffnet, Fehler in einem anderen Terminal zu beheben.

Dieser Funktion kann beim Aufruf ein spezieller Text als Parameter mitgegeben werden. Geschieht dies nicht, wird ein Standardtext ausgegeben.

```

171 <FailureNotice 171>≡ (153b)
function FailureNotice {
    # Called by PQImport CheckGitStatus RebasePQBranch PQMigration
    # You can call this function with a text as parameter (optional)

    if [ ! "$1" ]
    then
        echo "Failure"
        echo "Something went wrong!"
    else
        for i in $*
        do
            String=${String}" "$i
        done

        echo -e ${String}
    fi
    echo
    echo "Break for fixing it in another terminal"
    echo "After fixing press RETURN to go on!"
    read a
}

<PQImport 263>

```

33.4.3. Auswahl der Debian-Banches

Die Funktion *DebianBranches* ermittelt, welche einschlägigen Branches im Git-Repositorium vorhanden sind.

Hierzu wird zunächst die Funktion *IdentifyBranches* aufgerufen (Kapitel 32.5.2, Seite 154).

```
172a <DebianBranches 172a>≡ (174)
function DebianBranches {
    # Called by CreateNewBranch SelectBranch
    # selects the Debian branches
    IdentifyBranches
}
```

<DebianBranches1 172b>

Im Folgenden wird die Auswahl auf die lokalen Debian-Banches begrenzt.

```
172b <DebianBranches1 172b>≡ (172a)
## Trim branchlist
bl=$(echo $bl | sed 's/pristine-tar/ /')
bl=$(echo $bl | sed 's/upstream/ /')
bl=$(echo $bl | sed 's/HEAD/ /')
# bl=$(echo $bl | sed 's/remotes\origin\././ /g')
# bl=$(echo $bl | sed 's/remotes\salsa\././ /g')
# bl=$(echo $bl | sed 's/remotes\home\././ /g')
bl=$(echo $bl | sed 's/remotes\././ /g')
}
```

<CreateNewBranch (nicht definiert)>

33.4.4. Dialog zur Auswahl eines Branches

Die Anzeige dieses Dialoges erfolgt nur, wenn es mehrere Debian-Branches gibt. Der aktuelle Branch wird vorausgewählt.

Existiert nur ein oder kein Branch, geht es im Kapitel 33.4.7 (Seite 176) weiter.

```

173  <SelectBranch1 173>≡ (169)
      ## Create a radiolist with the branch names
      ba=($bl)
      i=1; slct=''
      set +e
      for element in ${ba[*]}
      do
          echo ${element} | grep 'x_' > /dev/null
          if [ $? -eq 0 ]
          then
              if [ "${element}" = "x_" ]
              then
                  continue
              else
                  ostr="on"
              fi
          else
              ostr="off"
          fi
          slct=${slct}' '$i' '$element}' '$ostr}' '
          i=$(expr $i + 1)
      done
      set -e

      if [ $#ba[@] -gt 1 ]
      then
          ## select branch
          branch=$(whiptail --title "Branch" --radiolist "Select:" \
          15 60 8 ${slct} --cancel-button "Task selection" 3>&2 2>&1 1>&3)
          if [ ! -z "${branch}" ]
          then
              set +e
              branch=$(expr ${branch} - 1)
              set -e
              bName=${ba[$branch]}
              bName=$(echo ${bName} | sed --expression='s/^x_//')
              ## checkout branch
              git checkout ${bName}
              ## Change config file -
              ## make selected branch to recent one
              ChangeEntry
          <SelectBranch3 175a>

```

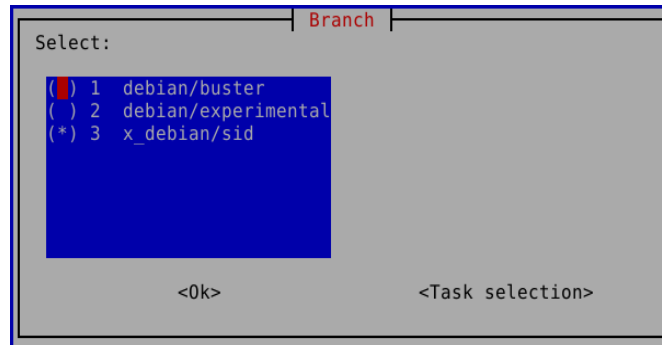


Abbildung 33.5.: Auswahl des Debian Zweiges.

Statt *debian-stable* wird der für diesen Git-Zweig gewählte Name angezeigt.

Ist dies nicht der Fall, wird mit dem Klick auf den Button *Task selection* die Aufgabenauswahl ausgewählt (Kapitel 33.5, Seite 177). Dort kann mit *Exit* auch das Programm verlassen werden. Mit *Create new branch* kann dort auch ein neuer Git-Zweig angelegt werden.

Mit *Ok* wird die Konfigurationsdatei nochmals mit *less* angezeigt.

33.4.5. Eintrag ändern

```

174 <ChangeEntry 174>≡
function ChangeEntry {
    # Called by CreateNewBranch SelectBranch
    set +e
    RecentBranchEntry=$(grep --count 'RecentBranch=' ${ConfigPath}${OrigName})
    set -e

    ## Change RecentBranch entry in config file
    if [ ${RecentBranchEntry} -eq 0 ]
    then
        echo "RecentBranch=${bName} >> ${ConfigPath}${OrigName}"
    else
        # ReplaceConfigLines needs two parameters:
        # name of the variable and new value
        ReplaceConfigLines 'RecentBranch' ${bName}
        # bName1=$(echo ${bName} | sed --expression='s/\//\\/g')
        # sed --in-place --expression=\
        # "s/RecentBranch=.*\/RecentBranch=${bName1}/g" \
        # ${ConfigPath}${OrigName}
    fi
    less --LINE-NUMBERS ${ConfigPath}${OrigName}

    ## Set variable
    RecentBranch=${bName}
    echo "Notice from ChangeEntry: The branch is "${RecentBranch} >> ${log}
}

<DebianBranches 172a>

```

```

175a  <SelectBranch3 175a>≡ (173)
      whiptail --title "This branch was selected" \
      --msgbox "${bName} was selected" 15 60
      echo "${bName} was selected" >> ${log}
      ParseConfig
      fi
  <SelectBranch4 176a>

```

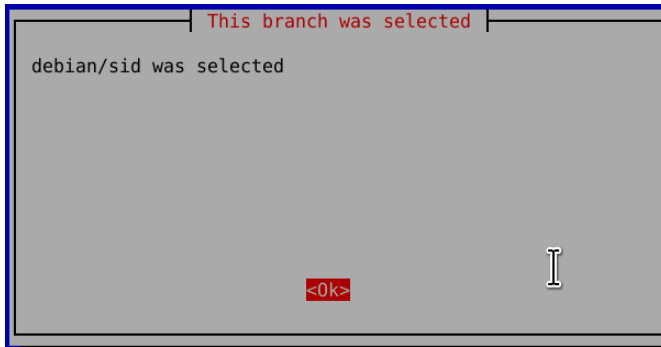


Abbildung 33.6.: Ausgewählter Debian Zweig.

In der Nutzeroberfläche geht es weiter mit der Aufgabenauswahl (Kapitel 33.5, Seite 177).

33.4.6. Konfiguration einlesen

Diese Funktion ordnet dem ausgewählten Git-Zweig die Debian-Distribution zu, für die das Paket gebaut wird. (Kapitel 37.3.4, Seite 307)

```

175b  <ParseConfig 175b>≡
      function ParseConfig {
          # Called by SelectBranch TaskSelect

          # Parse config file for Debian distribution of branch
          set +e
          vc=$(grep --count ${bName}_Dist ${ConfigPath}/${OrigName})
          set -e
          if [ $vc -ge 1 ]
          then
              Search4Dist
          else
              va="sid"
          fi
          RecentBranchD=${va}
          echo "Notice from ParseConfig: The distribution is "${RecentBranchD} >> ${log}
      }

  <SelectBranch 169>

```

33.4.7. Kein oder nur ein Branch existiert

```

176a  <SelectBranch4 176a>≡ (175a)
      elif [ ${#ba[@]} -eq 1 ]
      then
          whiptail --title "Only one branch" \
          --msgbox "There is only one Debian branch: ${ba[0]}" 15 60
      <SelectBranch6 176b>

```

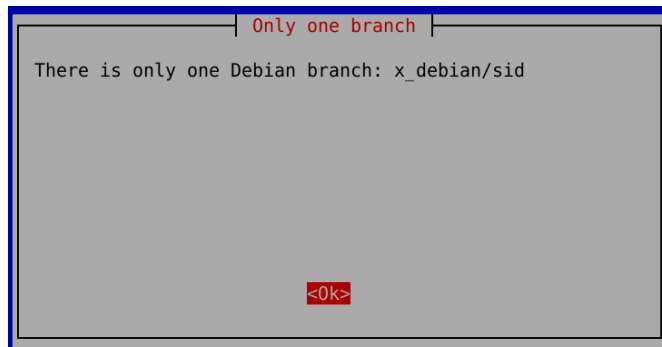


Abbildung 33.7.: Es gibt nur einen Git-Zweig

Gibt es einen Git-Zweig geht es mit der Aufgabenauswahl weiter (Kapitel 33.5, Seite 177). Andernfalls erfolgt ein Hinweis.

```

176b  <SelectBranch6 176b>≡ (176a)
      else
          whiptail --title "There is no branch" \
          --msgbox "There is no branch created.\nPlease build a new version." 15 60
      fi
  }

  <AddGitServer (nicht definiert)>

```

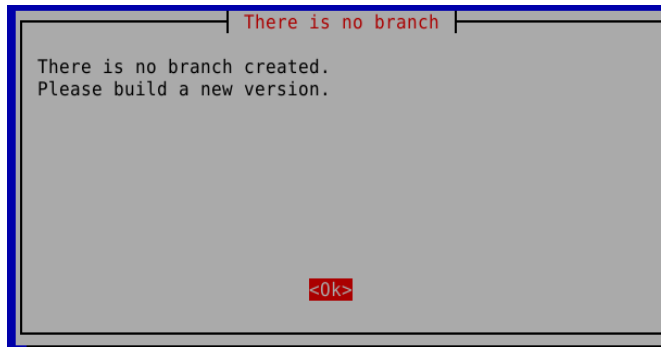


Abbildung 33.8.: Kein Zweig erstellt

33.5. Aufgabenauswahl

Nachdem die Konfigurationsdatei geladen und geprüft oder erstellt wurde, erscheint ein Menü zur Aufgabenauswahl.

Da das Programm modular aufgebaut ist, können die Aufgaben auch einzeln ausgewählt werden. Der Bauprozess kann abgebrochen und später wieder aufgenommen werden.

```

177 <CommonTasks 177>≡ (159)
    function CommonTasks {
        # Called by BuildApp TaskSelect

        Task=$(whiptail --title "Tasks:" \
            --radiolist "What do you like to do? " 17 60 9 \
                "2" "Build a new version of a package" off \
                "3" "Build a new debian revision" on \
                "4" "Rebuilding a revision" off \
                "5" "Running lintian and uscan" off \
                "6" "Uploading only (build again if necessary)" off \
                "7" "Create new branch" off \
                "8" "Select branch" off \
                "9" "Set name or IP of own git server" off \
                "10" "Create a debdiff" off \
            --cancel-button "Exit" 3>&2 2>&1 1>&3)

        if [ -z "${Task}" ]
        then
            exit
        fi
        TaskSelect
    }

    <AskOrigName 107b>

```

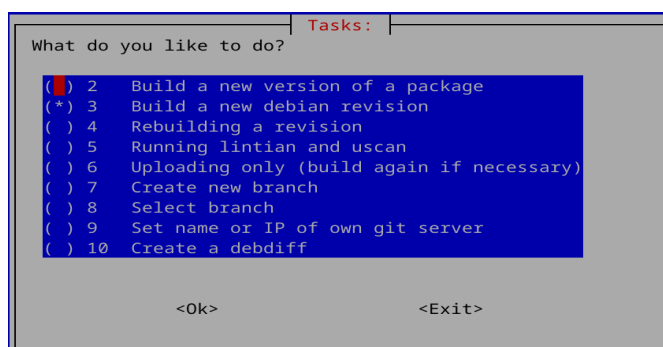


Abbildung 33.9.: Aufgabenauswahl.

Zur Auswahl stehen zunächst der Bau einer neuen Version (Kapitel 34, Seite 181) und einer neuen Revision. Als *default* geht es mit dem Bauen einer neuen Debian-Revision (Kapitel 35, Seite 225) weiter.

Soll eine neue Version gebaut werden, wird gefragt, ob zunächst Änderungen von *salsa.debian.org* heruntergeladen werden sollen (Kapitel 34.1, Seite 181).

In der Funktion *ClassicalOrUscan* (Kapitel 34.2, Seite 183) wird abgefragt, auf welche Weise das Herunterladen des Upstream-Quellcodes erfolgen.

Nach dem Bauen können die Pakete geprüft werden (Kapitel 40, Seite 327).

Der nächste Menüpunkt betrifft das Hochladen (Kapitel 41, Seite 343).

Ferner kann auch ein neuer Git-Zweig angelegt werden, beispielsweise auch für Backports, gearbeitet wird (Kapitel 44.1, Seite 375).

Sind mehrere Git-Zweige vorhanden, kann einer von Ihnen ausgewählt werden (Kapitel 33.4, Seite 169).

Schließlich kann man auch einen eigenen Git-Server in den Arbeitsablauf einbinden (Kapitel 44.2, Seite 376).

Die folgenden Zeilen des Programmskripts dienen der Ablaufsteuerung. Eine aufgerufene Funktion ändert an ihrem Ende die Variable *Task*, so dass eine der folgenden if-Klauseln zutrifft. Aus diesem Grund können die folgenden Zeilen nicht durch eine Case-Anweisung ersetzt werden.

Die Kette von If-Anweisungen gibt die Möglichkeit, eine Bedingung nach der anderen abzuarbeiten, da die aufgerufene Funktion an das Ende der If-Anweisung zurückkehrt, von der sie aufgerufen wurde. Eine Case-Anweisung wird mit dem Aufruf der abzuarbeitenden Funktion beendet.

Der Aufruf der entsprechenden Funktionen erfolgt in der genannten Weise wiederum durch die Funktion *TaskSelect*.

Mit dem folgenden Aufruf wird das Bauen einer neuen Version aufgerufen (Kapitel 34.1, Seite 181).

```
178 <TaskSelect3 178>≡ (134c)
    ## Common tasks
    rcts=0 # ReCall TaskSelect flag

    # Building a new version
    if [ $Task -eq 2 ]
    then
        if [ ${NoPull} -eq 0 ]
        then
            PullFromSalsa
        fi
        ClassicalOrUscan # and then download new version
```

```
fi
```

```
<TaskSelect4 179a>
```

Oder es geht weiter mit dem Bauen einer neuen Debian-Revision (Kapitel 35, Seite 225). Dies entspricht dem voreingestellten Wert.

```
179a <TaskSelect4 179a>≡ (178)
```

```
# Building a new revision
if [ $Task -eq 3 ]
then
    BuildNewRevision
fi

# Rebuilding a revision
if [ $Task -eq 4 ]
then
    ParseConfig
    AskDist
    SBuildOrPBuilder 0
    Task=5 # Running Tests
fi
```

```
<TaskSelect5 179b>
```

Nach dem Bauen einer neuen Revision ist diese zu testen. (Kapitel 40, Seite 327) weiter. Dies geschieht mittels *lintian* (Kapitel 40.3, Seite 330) und *uscan* (Kapitel 40.4, Seite 332).

```
179b <TaskSelect5 179b>≡ (179a)
```

```
# Running lintian and uscan
if [ $Task -eq 5 ]
then
    RunningTests
fi
```

```
<TaskSelect6 179c>
```

Mit dem nächsten Schritt wird die Funktion *PrepareUploading* aufgerufen (Kapitel 41, Seite 343). Darin wird zunächst überprüft, ob die Datei *debian/changelog* schon für eine Veröffentlichung vorbereitet ist. Andernfalls wird dies noch durchgeführt.

```
179c <TaskSelect6 179c>≡ (179b)
```

```
# Uploading the package
if [ $Task -eq 6 ]
then
    PrepareUploading
fi
```

```
<TaskSelect7 180>
```

Der folgende Abschnitt ist notwendig, um für eine abweichende Distribution zu bauen. Dies bezieht sich auf Debian-Backports, Debian-Proposed-Updates für Stable und eventuell für Oldstable. Gegebenenfalls wird dies auch benötigt, um für einen Ubuntu-Zweig ein Paket bereitzustellen. (Siehe auch Kapitel 39, Seite 325)

```

180 <TaskSelect7 180>≡ (179c)
    # Create new branch
    # (e.g. for backports or proposed-updates)
    if [ $Task -eq 7 ]
    then
        CreateNewBranch
        rcts=1
    fi

    # Select branch
    if [ $Task -eq 8 ]
    then
        SelectBranch
        rcts=1
    fi

    # Set name or IP of own git server
    if [ $Task -eq 9 ]
    then
        OwnServer
        rcts=1
    fi

    # Create a debdiff
    if [ $Task -eq 10 ]
    then
        DebDiff 0
        rcts=1
    fi

<TaskSelect9 (nicht definiert)>

```


34. Bauen einer neuen Version

In diesem Kapitel werden die Schritte beschrieben, die zur Erstellung des Debian-Quellcode-Paketes (**.orig.tar.(g/x)z*) (Kapitel 8, Seite 23) führen. Dies erfolgt nach dem Start des Programmskriptes (Kapitel 31.2, Seite 106).

Das Programmskript ermöglicht auf verschiedenen Weisen, den Upstream-Quellcode zu erlangen. Zwei Möglichkeiten wurden bereits beschrieben, nämlich das Klonen von *salsa.debian.org* (Kapitel 32.5, Seite 150) sowie der Import eines Debian-Quellcode-Paketes (Kapitel 32.6, Seite 156)

Zwei weitere Möglichkeiten werden im Folgenden beschrieben (Kapitel 34.2, Seite 183). Nach der Ausführung von *mk-origtar.gz* zum Erstellen des Debian-Quellcode-Paketes folgt die Integration des Quellcodes in das lokale Git-Repository (Kapitel 34.3.11, Seite 218).

Zunächst wird jedoch die Möglichkeit eröffnet, eventuelle Änderungen im Salsa-Repository des Paketes herunterzuladen, was manchmal sinnvoll ist.

34.1. Änderungen von *Salsa* herunterladen

Es kann besonders bei team-betreuten Paketen vorkommen, dass andere Team-Mitglieder Änderungen im Git-Repository auf *salsa.debian.org* bereitstellen. Für die weiteren Arbeiten an diesem Repository ist es nun zwingend erforderlich, diese Änderungen auch ins lokale Repository zu übernehmen.

Dies kann erfolgen mit

```
gbp pull --all salsa
```

Wenn dies zu Problemen führt, können die einzelnen Branches auch mit

```
git checkout <BranchName>
git pull salsa <BranchName>
```

aktualisiert werden.

181 $\langle PullFromSalsa\ 181 \rangle \equiv$ (185b)

```
function PullFromSalsa {
    # Called by TaskSelect

    cd ${GitPath}
    set +e
    if git remote --verbose | grep --quiet 'salsa'
    then
        if whiptail --title "Pull from Salsa?" \
            --yesno "Do you like to pull possible changes from salsa?" \
            --yes-button "Yes" --no-button "No" 15 60
        <PullFromSalsa1 182a>
```

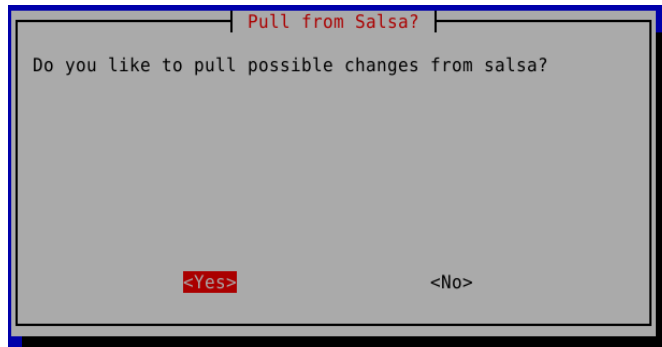


Abbildung 34.1.: Herunterladen von salsa.debian.org

Dazu wird dann noch das Passwort für den SSH-Schlüssel abgefragt, mit dem auf *salsa.debian.org* zugegriffen werden kann.

```
182a <PullFromSalsa1 182a>≡ (181)
    then
        echo "RecentBranch: "$(git branch) >> ${log}
        gbp pull --all salsa
        if [ $? -eq 0 ]
        then
            whiptail --title "Pulled successfully!" \
                --msgbox "'gbp pull --all salsa' successfully executed" 15 60
        <PullFromSalsa2 182b>
```

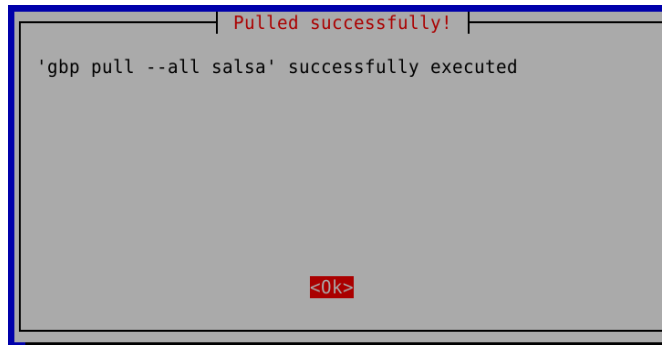


Abbildung 34.2.: Erfolgreich heruntergeladen von salsa.debian.org

```
182b <PullFromSalsa2 182b>≡ (182a)
    else
        FailureNotice
    fi
fi
fi
set -e
}
```

<CloneFromSalsa 150b>

Danach wird die Funktion *ClassicalOrUscan* (Kapitel 34.2, Seite 183) aufgerufen.

34.2. Werkzeuge zum Herunterladen der Upstream-Sourcen

Zum Bauen eines **Debian**-Paketes wird zunächst der Quellcode des Upstream-Projektes benötigt. Dieser kann auf verschiedene Arten erlangt werden.

Das Herunterladen des Quellcodes mit *wget* ist die klassische Methode (Kapitel 34.3, Seite 186). Dieser Weg ist beim ersten Bauen eines neuen Paketes zu wählen.

Es kommt vor, dass Upstream kein Quellcode-Archiv zur Verfügung stellt. Stattdessen findet man den Quellcode bei *Github* oder ähnlichen Hostern. Dies stellt besondere Anforderungen an den Betreuer eines **Debian**-Paketes.

Alleine schon, um *reproducible* bauen zu können, muss ein Tar-Archiv des Upstream-Quellcodes bereitgestellt werden. Dies muss ohne Netzzugriff auf Externe Ressourcen möglich sein.

Diese Methode ist auch zu wählen, wenn ein bestimmter Stand des Upstream-Codes aus einem Git-Repository (z.B. Github o.ä.) verwendet werden soll.

Existieren jedoch bereits eine Datei *debian/watch* und andere Dateien im Verzeichnis *debian/*, kann hierfür auch *uscan* (Kapitel 34.4, Seite 221) eingesetzt werden. Ein heruntergeladen mit *uscan* empfiehlt sich dann, wenn die Datei *debian/watch* einen Eintrag *wversionmangle=* enthält.

Kann *uscan* die neue Version nicht identifizieren, muss die neue Version manuell oder per *wget* bereitgestellt werden. Gleiches gilt, wenn *uscan* die neue Version zwar identifizieren kann, ein Herunterladen aber am Aufbau der Adresse zum Herunterladen scheitert.

Diese Möglichkeiten werden vom Programmskript alternativ angeboten.

Außerdem gibt es noch die Möglichkeit *get-orig-source* in *debian/rules* zu verwenden.

```
183 <ClassicalOrUscan 183>≡ (266a)
function ClassicalOrUscan {
    # Called by PullFromSalsa

    if [ ! -f ${GitPath}/debian/watch ]
    then
        BuildNewVersion
    return
    <ClassicalOrUscan1-1 184a>
```

4. Mai 2025

Es wird geprüft, ob in der Datei *debian/watch addons.thunderbird.net* oder *addons.mozilla.org* als Quelle aufgeführt werden. Von dort ist nämlich kein Herunterladen mit *uscan* möglich. Denn die Adresse zum Herunterladen enthält dort zusätzlich zur Versionbezeichnung eine nicht vorhersehbare ID.

184a `<ClassicalOrUscan1-1 184a>≡` (183)

```
else
  set +e
  # Download from Mozilla repos is not possible with uscan
  if grep --quiet "addons.thunderbird.net" ${GitPath}/debian/watch
  then
    whiptail --title "Thunderbird Repository" \
      --msgbox "From addons.thunderbird.net \nyou can't download with uscan" \
      15 60
    BuildNewVersion
    set -e
    return
  fi
```

`<ClassicalOrUscan2 184b>`

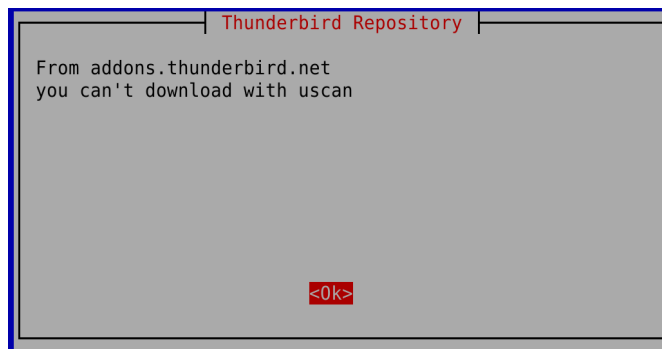


Abbildung 34.3.: Kein Herunterladen per *uscan* von *thunderbird.net*

184b `<ClassicalOrUscan2 184b>≡` (184a)

```
if grep --quiet "addons.mozilla.org" ${GitPath}/debian/watch
then
  whiptail --title "Mozilla Repository" \
    --msgbox "From addons.mozilla.org \nyou can't download with uscan" \
    15 60
  BuildNewVersion
  set -e
  return
fi
set -e
```

`<ClassicalOrUscan3 185a>`

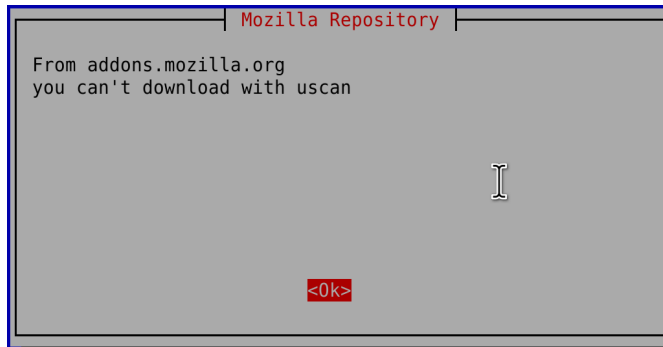


Abbildung 34.4.: Kein Herunterladen per uscan von mozilla.org

Ist kein Herunterladen mit *uscan* möglich, wird die Funktion *BuildNewVersion* zum Herunterladen auf klassische Weise aufgerufen (Kapitel 34.3.2, Seite 186)

Erscheint ein Herunterladen mit *uscan* möglich, wird der Benutzer gefragt, ob er auf „klassische“ Weise oder mit *uscan* die neue Version herunterladen möchte.

```
185a <ClassicalOrUscan3 185a>≡ (184b)
      NVTask=$(whiptail --title "Classical download or uscan" \
      --radiolist "How do you want to download the new version? " 17 60 9 \
      "0" "using the classical way" on \
      "1" "using uscan" off --cancel-button "Exit" 3>&2 2>&1 1>&3)
```

<ClassicalOrUscan5 185b>

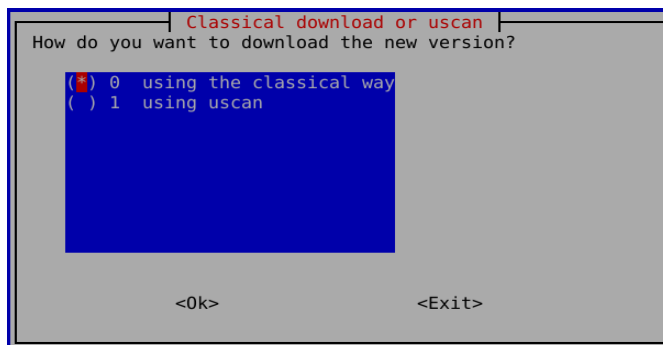


Abbildung 34.5.: Download - klassisch oder mit uscan

```
185b <ClassicalOrUscan5 185b>≡ (185a)
      if [ -z "${NVTask}" ]
      then
          exit
      fi
      case "$NVTask" in
          0) BuildNewVersion;;
          1) BuildWithUscan;;
      esac
  fi
}
```

<PullFromSalsa 181>

Existiert die Datei *debian/watch* nicht, muss der Benutzer die neue Version auf klassische Weise herunterladen.

Wird statt des klassischen Weges das Herunterladen mit *uscan* ausgewählt, geht es mit Kapitel 34.4 (Seite 221) weiter.

34.3. Herunterladen auf klassische Weise

In der Regel wird der Quellcode einer Software als Archiv bereitgestellt. Dazu stehen verschiedene Formate zur Verfügung. Die Verwendung derselben wird im Folgenden beschrieben.

34.3.1. Archiv-Formate

Für die Nutzung mit *git-buildpackage* wird zwingend ein Orig-Tar-Archiv als Quelle benötigt. Dieses darf nur die Formate **.tar.gz* oder **.tar.xz* haben. Das Orig-Tar-Archiv wird auch in das Debian-Archiv hochgeladen.

Zur Nutzung unter Linux wird meist ein **.tar.gz* bereitgestellt. Manchmal ist dies auch ein **.tar.xz*.

Für betriebssystemübergreifende Software wird der Quellcode gerne als Zip-Archiv bereitgestellt. Ein Zip-Archiv wird deshalb in ein *.tar.xz* umgepackt. Dazu gibt es das Tool *mk-origtargz* (Kapitel 34.3.6, Seite 205).

Zusätzlich kann in der zum Projekt gehörigen Datei *debian/gbp.conf* angegeben werden, welches Archiv-Format verwendet werden soll. Wird hier als Kompression *compression = xz* angegeben, muss auch ein **.tar.gz* in ein **.tar.xz* umgewandelt werden. Dies wird dann in der Datei *debian/README.source* dokumentiert (Kapitel 35.4.19, Seite 251).

Mit *mk-origtargz* können der Tarball der Originalautoren umbenannt, optional die Komprimierung geändert und unerwünschte Dateien entfernt werden.

34.3.2. Herunterladen des Quellcodes

Zunächst wird also der Quellcode heruntergeladen.

```
186 <BuildNewVersion 186>≡ (220)
    function BuildNewVersion {
        # Called by ClassicalOrUsan

        echo "Building a new version" >> ${log}

        UpstreamSourceName=$(whiptail --title "Name of the source" \
        --inputbox "Please insert the file name of the upstream source version\n \
        to be downloaded or copied (including version and suffix):\n" \
        --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)

    <BuildNewVersion3 187a>
```

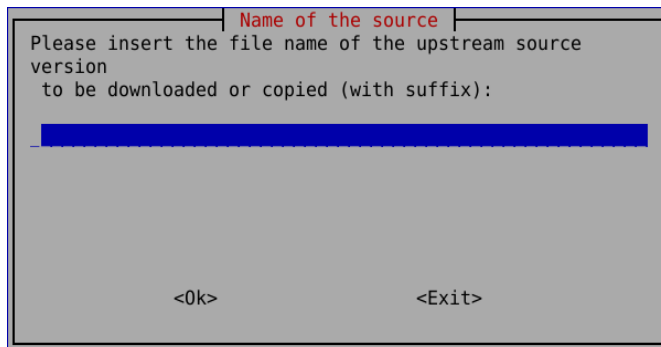


Abbildung 34.6.: Name des Quellcode-Paketes

Es ist der Name des Quellcode-Paketes einzugeben.

```
187a <BuildNewVersion3 187a>≡ (186)
      if [ -z "${UpstreamSourceName}" ]
      then
        exit
      fi
```

<BuildNewVersion4 187b>

Das Programm erledigt das Herunterladen der Upstream-Version. Ist diese bereits in den Projektpfad heruntergeladen worden, kann das Programm auch damit weiter arbeiten.

```
187b <BuildNewVersion4 187b>≡ (187a)
      cd ${PrjPath}
      if whiptail --title "Should the source be downloaded?" \
        --yesno "Should $UpstreamSourceName\n \
        be downloaded from the upstream page?" \
        --defaultno --yes-button "Yes" --no-button "No" 15 60
      <BuildNewVersion4-1 188a>
```

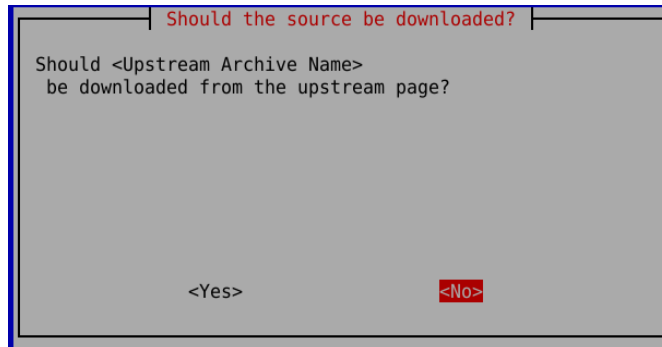


Abbildung 34.7.: Herunterladen (oder kopieren)?

Wird die Frage verneint, geht es mit Kopieren (Kapitel 34.3.2.2, Seite 190) weiter.

34.3.2.1. Herunterladen

188a `<BuildNewVersion4-1 188a>` ≡ (187b)

```

then
  if [ -z "${DownloadUrl}" ]
  then
    DownloadUrl=$(whiptail --title "Insert URL for download" \
      --inputbox "Please insert the complete\n \
        URL to download ${UpstreamSourceName}\n(with 'https://'\n \
          or so and the name of the archive):" \
      --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)
  fi

```

`<BuildNewVersion4-2 188b>`

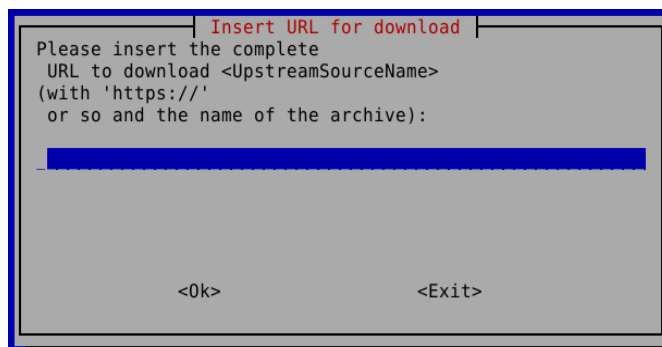


Abbildung 34.8.: Link für Download eingeben

188b `<BuildNewVersion4-2 188b>` ≡ (188a)

```

if [ -z "${DownloadUrl}" ]
then
  exit
else
  changeflag=1
fi

```

`<BuildNewVersion5 189a>`

Vorsorglich wird nachgefragt, ob die URL zum Herunterladen des Quellcodes korrekt ist.

```
189a  <BuildNewVersion5 189a>≡ (188b)
      if ! whiptail --title "DownloadUrl" \
        --yesno "The complete URL to download ${UpstreamSourceName} is\n \
        ${DownloadUrl}" --yes-button "Yes" --no-button "No" 15 60
<BuildNewVersion5-1 189b>
```

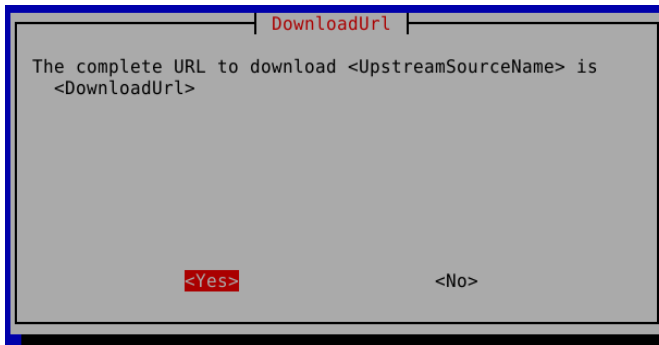


Abbildung 34.9.: Download-URL korrekt?

```
189b  <BuildNewVersion5-1 189b>≡ (189a)
      then
        DownloadUrl=$(whiptail --title "Complete URL" \
          --inputbox "Real complete URL to download ${UpstreamSourceName}\n \
          (with 'https://' or so and the name of the archive):" \
          --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)
      fi
<BuildNewVersion5-2 189c>
```

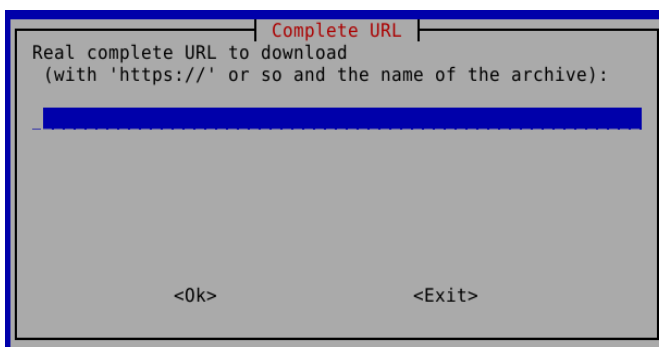


Abbildung 34.10.: Korrekte Download-URL

```
189c  <BuildNewVersion5-2 189c>≡ (189b)
      if [ -z "${DownloadUrl}" ]
      then
        exit
      else
        changeflag=1
      fi
<BuildNewVersion6 190a>
```

4. Mai 2025

Die neue URL zum Herunterladen des Quellcodes wird in die Konfigurationsdatei eingetragen. Danach erfolgt das Herunterladen mittels *wget*.

Dann wird die Möglichkeit eröffnet, auch die Signaturdatei herunterzuladen und zu prüfen (Kapitel 34.3.7, Seite 208).

```
190a <BuildNewVersion6 190a>≡ (189c)
      # Write download URL into config file
      if [ $changeflag -eq 1 ]
      then
          ReplaceConfigLines 'DownloadUrl' ${DownloadUrl}
          changeflag=0
      fi

      # getting sources using wget
      wget --verbose $DownloadUrl &&
      echo -e "The sources were pulled from\n${DownloadUrl}\n \
      by wget." >> ${log}

      if whiptail --title ".asc file?" \
      --yesno "Do you want to download an .asc file, too?" \
      --yes-button "Yes" --no-button "No" 15 60
      then
          DownloadAscFile
      fi
<BuildNewVersion6-1 190b>
```

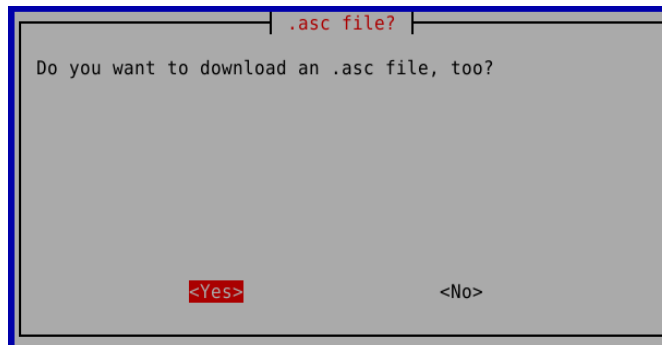


Abbildung 34.11.: *.asc Datei herunterladen

34.3.2.2. Kopieren des Quellarchivs

Das Programmskript teilt mit, wohin die Upstream-Version kopiert werden soll.

```
190b <BuildNewVersion6-1 190b>≡ (190a)
      else
          whiptail --title "Please copy the source code now" \
          --msgbox "Please copy ${UpstreamSourceName} to ${PrjPath}!" 15 60
<BuildNewVersion6-2 191a>
```

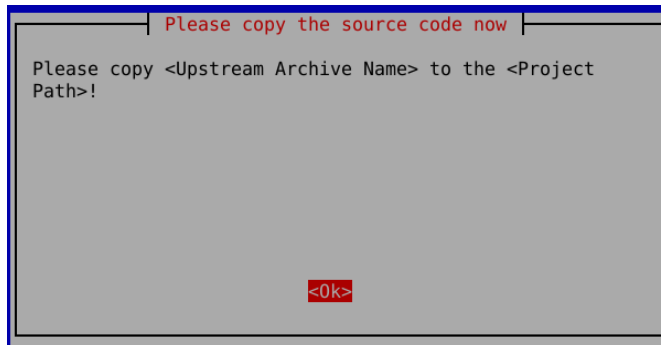


Abbildung 34.12.: Aufforderung zum Kopieren

Das Programmskript fragt, ob das Kopieren erledigt wurde. Wird diese Frage verneint, beendet sich das Programmskript.

```
191a <BuildNewVersion6-2 191a>≡ (190b)
      if ! whiptail --title "Copy finished?" \
        --yesno "Was ${UpstreamSourceName} copied to ${PrjPath}?" \
        --yes-button "Yes" --no-button "No" 15 60
<BuildNewVersion6-3 191b>
```

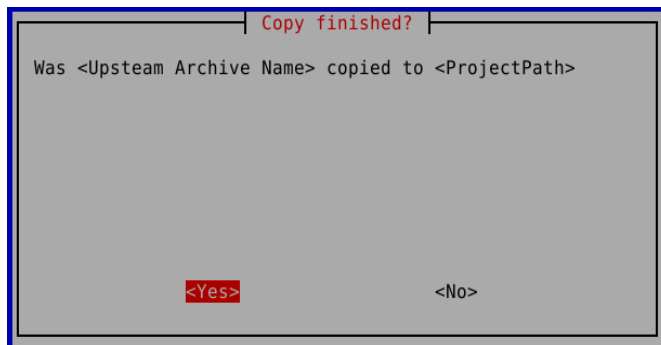


Abbildung 34.13.: Kopieren erledigt?

```
191b <BuildNewVersion6-3 191b>≡ (191a)
      then
        echo "Exit" >> ${log}
        whiptail --title "Bye" --msgbox "Bye" 15 60
        exit
      fi
<BuildNewVersion6-4 192a>
```

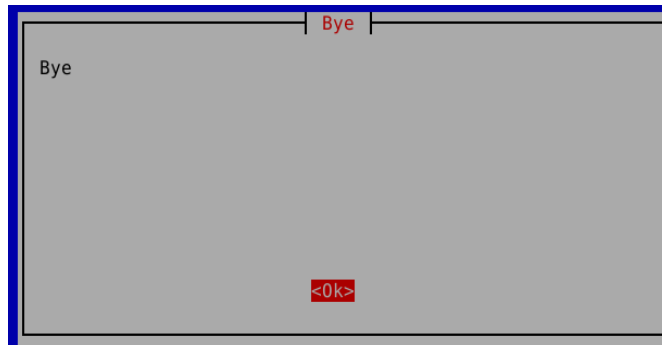


Abbildung 34.14.: Programm beenden

192a `<BuildNewVersion6-4 192a>≡` (191b)
`fi`

`# Identify the type of the upstream archive by suffix`
`CutSuffix`

`<BuildNewVersion7 193b>`

Nun wird die Komprimierung und die Versionsnummer (s. Kapitel 34.3.4 (Seite 193) ermittelt. Dies wird auch insgesamt als *Suffix* bezeichnet. Ist hier alles in Ordnung geht es im Kapitel 34.3.5 (Seite 197) weiter.

34.3.3. Komprimierung ermitteln

Welche Komprimierung *mk-origtargz* automatisch wählt, hängt vom Dateityp des Upstream-Archivs ab. Dabei wird die Dateinamenserweiterung mit einer Liste sinnvoller Dateitypen verglichen.

192b `<CutSuffix 192b>≡` (156b)
`function CutSuffix {`

`# Called by BuildNewVersion`

`# List of reasonable suffixes`

`typea=('.tar.gz' '.tar.xz' '.tgz' '.zip' '.oxt' '.xpi'\`
`' .jar' '.tar.bz2' 'tar.lzma')`

`<CutSuffix1 193a>`

Die Dateitypen *.oxt*, *.xpi* und *.jar* beschreiben allesamt *.zip*-Archive.

```
193a  <CutSuffix1 193a>≡ (192b)
      KnownTyp=0
      set +e
      for element in ${typea[*]}
      do
          if echo ${UpstreamSourceName} | grep ${element} > /dev/null
          then
              echo "Notice from CutSuffix: The suffix of the upstream \
file is "${element}" >> ${log}
              UpstreamSuffix=${element}
              RecentUpstreamSuffix=$(echo ${UpstreamSuffix} | sed --expression s/^.//)
              ReplaceConfigLines 'RecentUpstreamSuffix' ${UpstreamSuffix}
              KnownTyp=1
          fi
      done
      set -e

      if [ ${KnownTyp} -ne 1 ]
      then
          echo "Notice from CutSuffix: Unknown suffix" >> ${log}
          if ! whiptail --title "Unknown suffix" \
--yesno "The suffix of ${UpstreamSourceName} is not listed.\n \
Continue anyway?" --defaultno --yes-button "Yes" \
--no-button "No" 15 60
          then
              exit
          fi
      fi
  }
```

<Name2Version 194a>

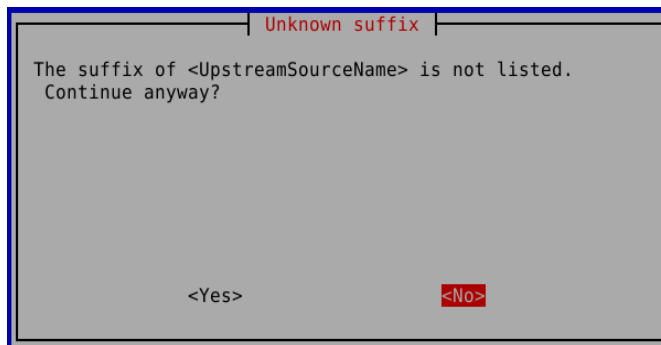


Abbildung 34.15.: Unbekannte Endung

34.3.4. Upstream-Version ermitteln

```
193b  <BuildNewVersion7 193b>≡ (192a)
      # Identify the upstream version number
      Name2Version

      <BuildNewVersion8 194b>
```

Die Funktion *Name2Version* versucht aus dem Namen des Upstream-Paketes die Versionsbezeichnung zu extrahieren.

```
194a <Name2Version 194a>≡ (193a)
function Name2Version {
    # Called by BuildNewVersion
    # Extracts version from upstream archive name
    Suffix=$(echo ${UpstreamSuffix} | sed --expression='s/\./\\./g')
    Version1=$(echo ${UpstreamSourceName} | sed --expression="s/${Suffix}$//" | \
sed --expression="s/.*${SourceName}//gI" | \
sed --expression='s/-//' | sed --expression='s/v//')
    if [ -z ${Version1} ]
    then
        Version1="0.0.0" # Default value
    fi
}
```

<GbpConfIntegration 210c>

Misslingt dies, wird die Versionsbezeichnung auf *0.0.0* gesetzt.

Die ermittelte (oder nicht ermittelte) Versionsbezeichnung wird dem Nutzer angezeigt. Der Nutzer hat zu entscheiden, ob die angezeigte Versionsbezeichnung die korrekte ist. Zu beachten ist das Versionierungsschema (Kapitel 11.2, Seite 35).

```
194b <BuildNewVersion8 194b>≡ (193b)
if ! whiptail --title "Version" \
--yesno "You want to build version ${Version1}.\n\n \
It must be a correct version number - not more!" \
--yes-button "Yes" --no-button "No" 15 60
<BuildNewVersion9 195a>
```

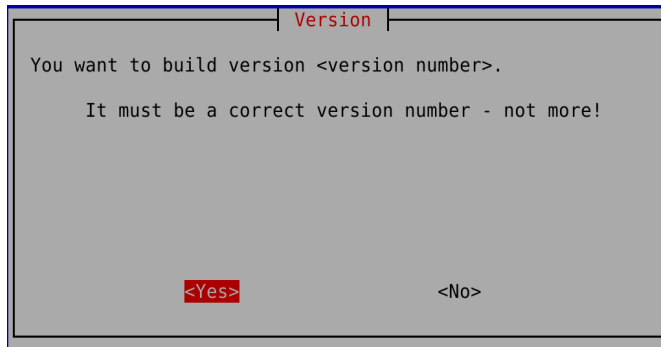


Abbildung 34.16.: Ist dies die korrekte Version?

Ist die Versionsbezeichnung korrekt, wird die Funktion *ExcludeFiles* aufgerufen (s. Kapitel 34.3.5 Seite 197)

Es kann vorkommen, dass das Programmskript die korrekte Versionsbezeichnung nicht ermitteln kann. Dann kann in der folgenden Dialogbox die Versionsbezeichnung eingegeben werden. Damit sie auch weiterverarbeitet werden kann, darf die Versionsbezeichnung neben den Ziffern nur Punkte enthalten. Sie darf ferner die Angabe enthalten, ob es sich um einen *Release-Kandidaten*, eine *Beta-* oder *Alpha-Version* handelt. Zulässig ist auch die Angabe, ob es sich um einen bestimmten *Commit* aus dem *Git*-Repositorium handelt.

```
195a <BuildNewVersion9 195a>≡ (194b)
      then
```

```
      Version1=$(whiptail --title "Version" \
        --inputbox "Name of the upstream version: ${UpstreamSourceName}\n \
        Which version (without repack identifiers and without revision)\n \
        of the package ${SourceName} should be built?" \
        --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
```

```
<BuildNewVersion10 195b>
```

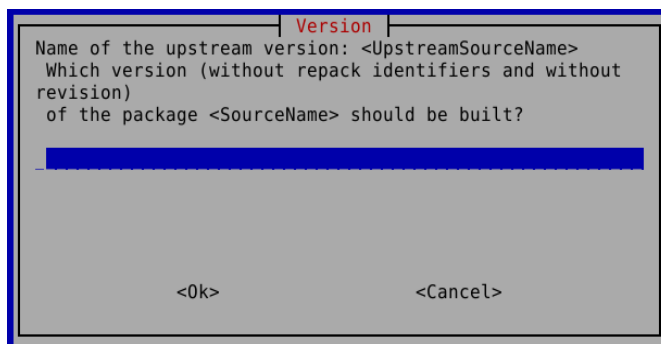


Abbildung 34.17.: Welche Version soll gebaut werden?

```
195b <BuildNewVersion10 195b>≡ (195a)
```

```
      if ! whiptail --title "Version" \
        --yesno "Do you really want to build version ${Version1}." \
        --yes-button "Yes" --no-button "Exit" 15 60
```

```
<BuildNewVersion11 196a>
```

4. Mai 2025

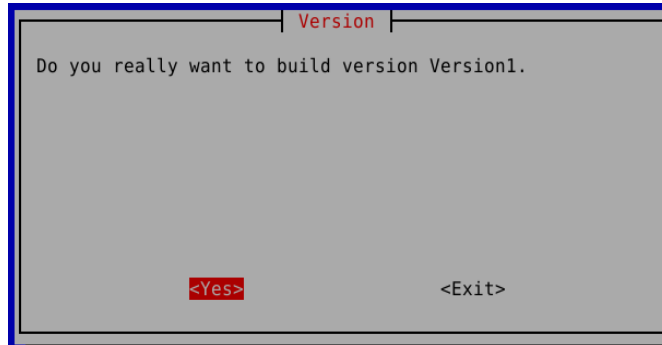


Abbildung 34.18.: Wird korrekte Version gebaut?

```
196a <BuildNewVersion11 196a>≡ (195b)
      then
        echo "Exit" >> ${log}
        whiptail --title "Bye" --msgbox "Bye" 15 60
        exit
      fi
<BuildNewVersion11-1 196b>
```

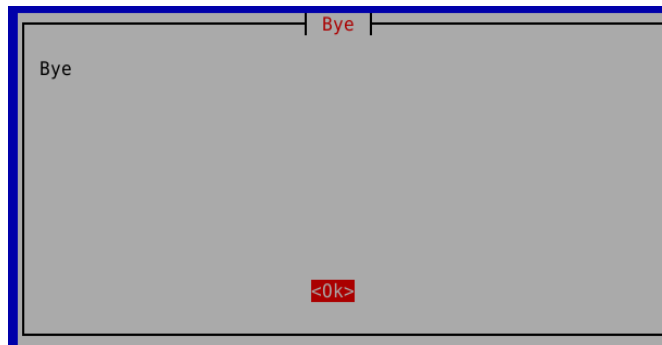


Abbildung 34.19.: Programm beenden

Die Funktion *ExcludeFiles* dient dazu, Dateien aus dem Upstream-Archiv von der Übernahme in die Datei **.orig.tar.xz* ausschließen.

Sollen keine Dateien ausgeschlossen werden, geht es mit Kapitel 34.3.6 (Seite 205)

```
196b <BuildNewVersion11-1 196b>≡ (196a)
      fi

      ExcludeFiles

      echo "Version "${Version1}${ESuffix}" of the package "${PackName}" \
      should be built." >> ${log}

<BuildNewVersion12 205b>
```


34.3.5. Dateien aus Upstream-Archiv ausschließen

Bevor *mk-origtargz* aufgerufen wird, ermöglicht das Programmskript einzelne Quellcode-Dateien von der Aufnahme in das *orig*-Archiv auszuschließen.

```
197a <ExcludeFiles 197a>≡ (200)
      function ExcludeFiles {
          # Called by BuildNewVersion
```

```
<ExcludeFiles1 197b>
```

Anzugeben ist, dass Dateien auszuschließen sind. Dazu ermittelt das Programmskript, woher die Informationen zum Ausschluss von Dateien erfolgen sollen. Dies ist hier die Datei *debian/copyright*.

Nun wird geprüft, ob eine Datei *debian/copyright* existiert.

```
197b <ExcludeFiles1 197b>≡ (197a)
      # Checks whether debian/copyright contains a section Files-Excluded
      gitflag=0
      exflag=0
      crflag=0
      if [ -f ${GitPath}/debian/copyright ]
      then
```

```
<ExcludeFiles2 197c>
```

Sodann wird geprüft, ob sie den Ausdruck *Files-Excluded* enthält. In diesem Fall wird abgefragt, ob die Datei *debian/copyright* editiert werden soll.

```
197c <ExcludeFiles2 197c>≡ (197b)
      crflag=1
      set +e
      grep 'Files-Excluded' ${GitPath}/debian/copyright > /dev/null
      if [ $? -eq 0 ]
      then
          exflag=1
          whiptail --title "Copyright file contains Files-Excluded" \
          --msgbox "debian/copyright contains section Files-Excluded." 15 60
          less --LINE-NUMBERS ${GitPath}/debian/copyright
      fi
      set -e
  fi
```

```
<ExcludeFiles3 198a>
```

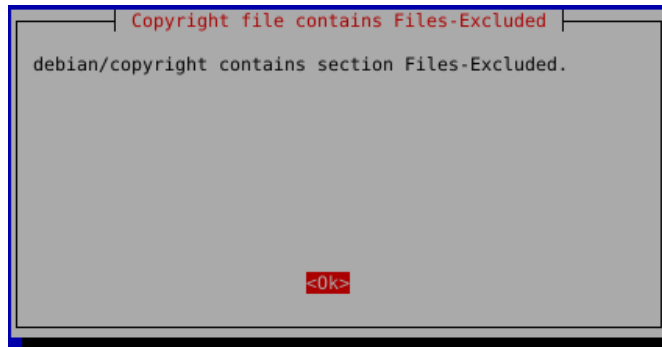


Abbildung 34.20.: Datei debian/copyright enthält Abschnitt Files-Excluded

```
198a <ExcludeFiles3 198a>≡ (197c)
      if whiptail --title "Exclude files from upstream source" \
        --yesno "Do you want to exclude files from upstream source?" \
        --defaultno --yes-button "Yes" --no-button "No" 15 60
      <ExcludeFiles3-1 198b>
```

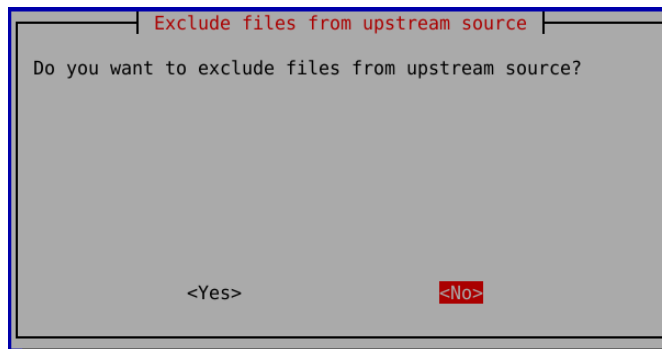


Abbildung 34.21.: Dateien ausschließen

Wird kein Ausschluss von Dateien benötigt, wird im Hintergrund *mk-origtargz* ausgeführt. (Kapitel 34.3.6, Seite 205). Dann geht es in Kapitel 34.3.8, (Seite 209) weiter.

```
198b <ExcludeFiles3-1 198b>≡ (198a)
      then
        if [ $crflag -eq 1 ]
        then
          if whiptail --title "Copyright file existst" \
            --yesno "debian/copyright exists.\nDo you want to edit it?" \
            --yes-button "Yes" --no-button "No" 15 60
          <ExcludeFiles3-2 199a>
```

Abbildung 34.22.: Soll `debian/copyright` editiert werden?

Wenn die Frage bejaht wird, wird die Datei `debian/copyright` im Editor geöffnet.

```
199a <ExcludeFiles3-2 199a>≡ (198b)
    then
        gitflag=1
        nano --linenumbers --mouse --softwrap ${GitPath}/debian/copyright
    fi
    AddOpt=" --copyright-file "${SourceName}"/debian/copyright"
```

<ExcludeFiles4 199b>

Nachdem die Datei `debian/copyright` editiert wurde oder wenn sie nicht editiert werden soll, geht es mit der Frage nach dem Suffix zur Anzeige des Ausschlussgrundes weiter (Seite 201).

Wenn die Datei `debian/copyright` nicht existiert, wird die Funktion `SpecialExcludeFile` aufgerufen. In dieser Funktion wird, sofern nicht - wie im vorliegenden Fall - die Angaben in der Datei `debian/copyright` verwandt werden, nach einer speziellen Datei gefragt, die die Namen der auszuschließenden Dateien im Format DEP-5¹ enthält.

```
199b <ExcludeFiles4 199b>≡ (199a)
    else
        SpecialExcludeFile
    fi
```

<ExcludeFiles5 201>

Steht schon vor der Einreichung des Paketes zur `New Queue` fest, dass Dateien auszuschließen sind, existiert zu diesem Zeitpunkt noch keine Datei `debian/copyright`. Es bietet sich dann an, die auszuschließenden Dateien in einer separaten Datei aufzulisten.

Die Funktion `ExcludeFiles` ruft hierfür die folgende Funktion auf. In dieser Funktion wird nach einer speziellen Datei gefragt, die die Namen der auszuschließenden Dateien im Format DEP-5 enthält.

```
199c <SpecialExcludeFile 199c>≡ (207)
function SpecialExcludeFile {
    # Called by ExcludeFiles
    if [ -z "${ExcludeFile}" ]
    then
        ExcludeFile=$(whiptail --title "Name of exclude file" \
            --inputbox "Please insert name of the exclude file:" \
            --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)
```

<SpecialExcludeFile1 200>

¹DEP-5[19]

4. Mai 2025

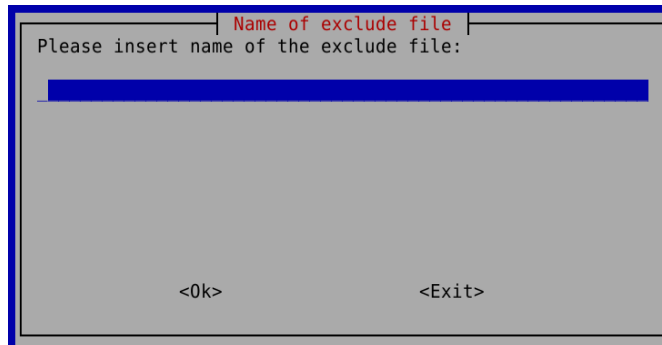


Abbildung 34.23.: Name der Datei mit den auszuschließenden Dateien eingeben.

```
200 <SpecialExcludeFile1 200>≡ (199c)
    if [ -z "${ExcludeFile}" ]
    then
        exit
    fi
    echo 'ExcludeFile=${ExcludeFile} >> ${ConfigPath}${OrigName}'
else
    whiptail --title "Exclude file name" \
    --msgbox "The name of the exlude file is ${ExcludeFile}" \
    15 60
fi
AddOpt="--copyright-file "${ExcludeFile}"
}
```

<ExcludeFiles 197a>

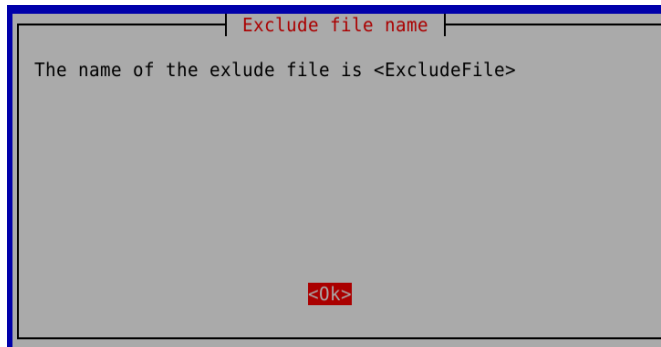


Abbildung 34.24.: Name der Datei mit den auszuschließenden Dateien.

Dann erfragt das Programmskript den Anhang zur Ergänzung der Upstream-Versionsbezeichnung.

Oft ist hier als Anhang *+dfsg* zu wählen, um den Grund des Ausschlusses zu dokumentieren (s.a Kapitel 10.4.1.3, Seite 32)

```

201  <ExcludeFiles5 201>≡ (199b)
      ESuffixN=$(whiptail --title "Suffix:" \
        --radiolist "Please choose the suffix: " \
        --cancel-button "Cancel" 15 60 4 \
        "0" "+ds" off \
        "1" "+dfsg" on \
        "2" "other" off 3>&2 2>&1 1>&3)
      if [ ${ESuffixN} -eq 1 ]
      then
        ESuffix="+dfsg"
      elif [ ${ESuffixN} -eq 0 ]
      then
        ESuffix="+ds"
  <ExcludeFile6 202a>

```

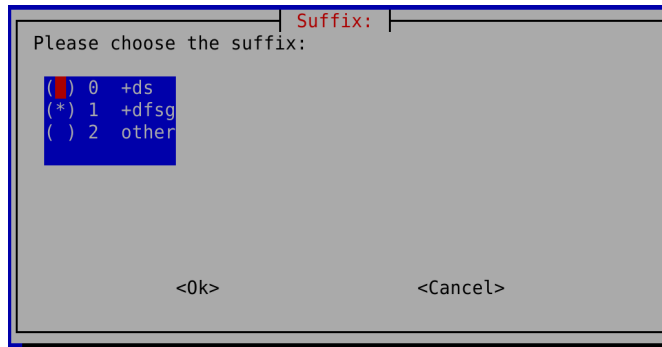


Abbildung 34.25.: Suffix für den Ausschluss von Dateien

Wenn ein anderes Suffix als die beiden vorgeschlagenen verwendet werden soll, ist dies manuell hinzuzufügen. Dieser Eintrag muss auch entsprechend in der Datei *debian/watch* ergänzt werden. (s.a Kapitel 35.4.7, Seite 239)

```
202a <ExcludeFile6 202a>≡ (201)
      else
        ESuffix=$(whiptail --title "Enter suffix" \
          --inputbox "Please insert the suffix:" \
          --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
      fi
<ExcludeFile7 202b>
```

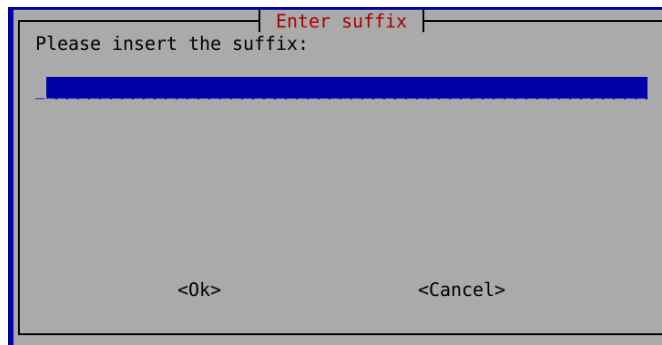


Abbildung 34.26.: Eigener Suffix für den Ausschluss von Dateien

```
202b <ExcludeFile7 202b>≡ (202a)
      if [ -z "${ESuffix}" ]
      then
        whiptail --title "Warning!" \
          --msgbox "You repacked the upstream source and\n\
          do not want to use a repack suffix." 15 60
        if [ -n "${RecentRepackSuffix}" ]
        then
          # Remove suffix from config file
          sed --in-place \
            --expression="s/RecentRepackSuffix=.*//g" \
            ${ConfigPath}${OrigName}
        fi
      else
<ExcludeFile8 203a>
```

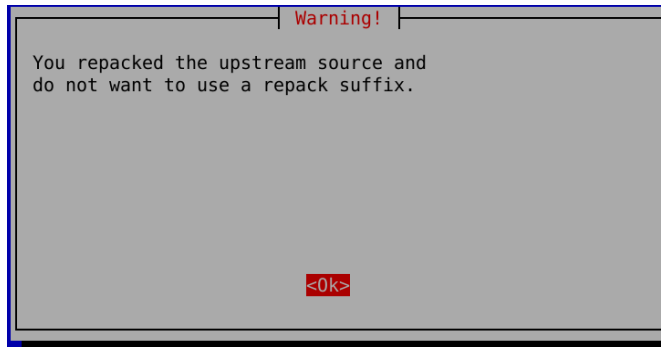


Abbildung 34.27.: Warnung! - Kein Suffix angegeben

In manchen Fällen kann ein Pluszeichen (+) Probleme vor allem beim Bauen von Java-Paketen verursachen. Manchmal ist es auch notwendig, dass kein Suffix hinzugefügt werden kann. Die Dokumentation solcher Vorgänge kann dann in der Datei *README.source* (Kapitel 35.4.19, Seite 251) erfolgen.

```
203a  <ExcludeFile8 203a>≡ (202b)
      # Insert suffix into config file
      if [ -z "${RecentRepackSuffix}" ]
      then
        echo "RecentRepackSuffix=${ESuffix} >> ${ConfigPath}${OrigName}
      else
        sed --in-place \
          --expression="s/RecentRepackSuffix=.*\/RecentRepackSuffix=${ESuffix}/g" \
            ${ConfigPath}${OrigName}
      fi

      RecentRepackSuffix=${ESuffix}
      AddOpt=${AddOpt}" --repack-suffix "${ESuffix}
    fi
  <ExcludeFiles10 203b>
```

Die Kompression des **.orig.tar.**-Archives wird in der Variable *suffix* hinterlegt. Er wird auf *.tar.xz* festgelegt.

```
203b  <ExcludeFiles10 203b>≡ (203a)
      whiptail --title "Option(s) for mk-origtargz:" \
        --msgbox "\n${AddOpt}" 15 60
    else
      AddOpt=""
      ESuffix=""
    <ExcludeFiles12 204>
```

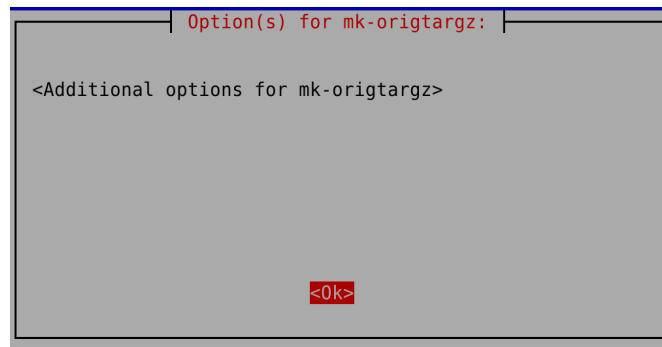
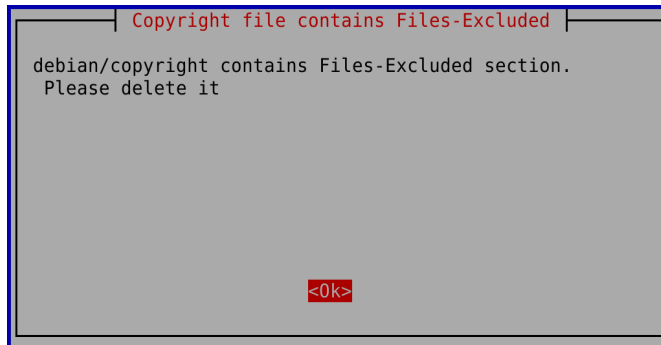


Abbildung 34.28.: Zusätzliche Optionen für *mk-origtargz*

Sollen keine Dateien ausgeschlossen werden, enthält aber die Datei *debian/copyright* einen Abschnitt *Files-Excluded:*, so ist dieser (manuell) zu entfernen.

```
204 <ExcludeFiles12 204>≡ (203b)
    if [ $exflag -eq 1 ]
    then
        gitflag=1
        whiptail --title "Copyright file contains Files-Excluded" \
        --msgbox "debian/copyright contains Files-Excluded section.\n \
        Please delete it" 15 60
        nano --linenumbers --mouse \
        --softwrap ${GitPath}/debian/copyright
    fi
fi

<ExcludeFiles15 205a>
```


Abbildung 34.29.: *debian/copyright* enthält auszuschließende Dateien

Wurde im Zusammenhang mit dem Ausschluss von Dateien die Datei *debian/copyright* bearbeitet, erfolgt ein entsprechender *commit*.

```
205a <ExcludeFiles15 205a>≡ (204)
    if [ $gitflag -eq 1 ]
    then
        set +e
        git add debian/copyright
        git commit -am "Changed debian/copyright"
        set -e
    fi
}
```

<CheckSignature 209a>

34.3.6. Debian-Quellcode-Datei erzeugen

Das Skript führt dann die Funktion *BuildNewVersion* weiter aus und übergibt dem Programm *mk-origtargz* die zum Ausschluss notwendigen Parameter. (Referenz auf diese Stelle in der anderen Funktion)

Auf diese Weise wird ein neuer Orig-Tarball mit *mk-origtargz* ohne die auszuschließenden Dateien aus dem bisherige *.tar.gz erstellt und dessen Inhalt mit *gbp import-orig* in das vorhandene Git-Repository eingefügt.

```
205b <BuildNewVersion12 205b>≡ (196b)
    # Creating orig file using mk-origtargz
    if [ -z ${Version1} ]
    then
        whiptail --title "No version number!" \
        --msgbox "No version - no *.orig.tar.gz! Bye!" \
        --ok-button "Exit" 15 60
        exit
    fi
    echo "mk-origtargz --package "${SourceName}" \
    --version "${Version1}${AddOpt}" "${UpstreamSourceName}" >> ${log}
    mk-origtargz --package ${SourceName} \
    --version ${Version1}${AddOpt} ${UpstreamSourceName} 2>> ${log}
```

<BuildNewVersion13 206a>

4. Mai 2025



Abbildung 34.30.: Keine Versionsnummer – kein *mk-origtargz*

```
206a <BuildNewVersion13 206a>≡ (205b)
      if [ $? -eq 0 ]
      then
          echo "orig file was created by mk-origtargz" >> ${log}
          Version1=${Version1}${ESuffix}
      else
          echo "mk-origtargz failed" >> ${log}
          whiptail --title "Fatal error" \
          --msgbox "mk-origtargz failed" 15 60
          exit
      fi
```

<BuildNewVersion14 206b>

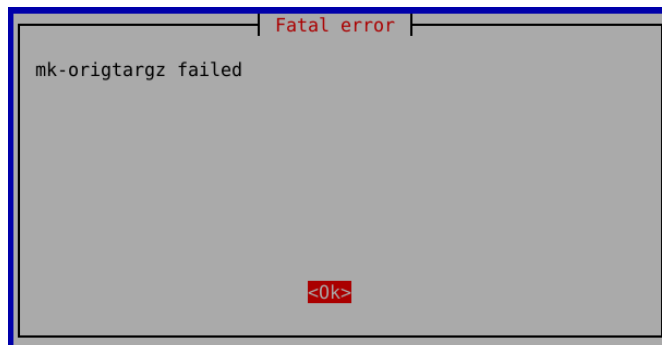


Abbildung 34.31.: *mk-origtargz* gescheitert

```
206b <BuildNewVersion14 206b>≡ (206a)
      Link2File
      SearchGbpConf

      cd ${GitPath}

      DebianBranch4Import
      <BuildNewVersion15 218a>
```

Wenn die Variable *RecentBranch* nicht existiert oder leer ist, wird ihr der Wert *debian/sid* zugeordnet und ein entsprechender Eintrag in der Konfigurationsdatei vorgenommen.

```
207  <DebianBranch4Import 207>≡ (301b)
      function DebianBranch4Import {
          # Called by BuildNewVersion # Makes sure RecentBranch contains value
          set +e
          RecentBranch=$(grep 'RecentBranch=' ${ConfigPath}${OrigName})
          RecentBranch=$(echo ${RecentBranch} | sed --expression='s/RecentBranch=//')
          set -e
          if [ -z "${RecentBranch}" ]
          then
              RecentBranch="debian/sid"
              changeflag=1
              whiptail --title "Set RecentBranch" \
                --msgbox "Set RecentBranch to ${RecentBranch}" 15 60
          fi

          if [ $changeflag -eq 1 ]
          then
              echo 'RecentBranch=${RecentBranch}' >> ${ConfigPath}${OrigName}
          fi
          echo "Notice from DebianBranch4Import: \
            The branch is ${RecentBranch} >> ${log}"
          changeflag=0
      }
```

<SpecialExcludeFile 199c>

34.3.7. Signatur prüfen

Manche Projekte veröffentlichen neben dem Quellcodepaket auch eine Signaturdatei. Das Skript kann diese herunterladen und eine kryptografische Prüfung durchführen. Erwartet wird eine Signaturdatei im *asc*-Format.

34.3.7.1. Signatur-Datei herunterladen

```

208  <DownloadAscFile 208>≡ (209a)
      function DownloadAscFile {
          # Called by BuildNewVersion and itself

          cd ${PrjPath}

          AscFileURL=$(whiptail --title "URL of .asc file" \
            --inputbox "URL and name (with suffix)\nof the .asc file:" \
            --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)

          if [ $? -eq 1 ]
          then
              return 1
          fi

          if [ -z "${AscFileURL}" ]
          then
              echo -e "URL and name (with suffix)\nof the .asc file:"
              read AscFileURL
          fi

          if [ -n "${AscFileURL}" ]
          then
              # getting -asc file using wget
              wget --version ${AscFileURL} &>> ${log}
              if [ $? -eq 0 ]
              then
                  echo -e "The .asc file was pulled from\n${AscFileURL}\n \
                    by wget." >> ${log}
                  whiptail --title "Download successful" \
                    --msgbox "${AscFileURL} was downloaded." 15 60
                  if [ -r ${UpstreamSourceName}.asc ]
                  then
                      CheckSignature
                  else
                      whiptail --title "There is something wrong!" \
                        --msgbox "Maybe you has downloaded the wrong .asc file." 15 60
                      DownloadAscFile
                  fi
              else
                  DownloadAscFile
              fi
          fi
      }

      <Link2File 209b>

```

34.3.7.2. Prüfung der Signatur

```

209a  <CheckSignature 209a>≡ (205a)
      function CheckSignature {
        # Called by DownloadAscFile
        gpg --verify ${UpstreamSourceName}.asc >> ${log}

        if [ $? -ne 0 ]
        then
          tail --lines=5 ${log}
          read x
        else
          whiptail --title "Check successfull!" --msgbox "gpg \
            --verify has been successfull" 15 60
        fi
      }

      <DownloadAscFile 208>

```

34.3.8. Link durch Kopie ersetzen

Das Standardverhalten von *mk-origtargz* (Kapitel 34.3.6, Seite 205) ist einen symbolischen Verweis auf die Originaldatei zu erzeugen, wenn diese unverändert übernommen wird.

Das Programmskript ersetzt diesen Verweis gegebenenfalls durch eine entsprechende Datei.

```

209b  <Link2File 209b>≡ (208)
      function Link2File {
        # Called by BuildNewVersion
        echo "Version: "${Version1} >> ${log}
        set +e
        OrigLinkNr=$(ls -la ${PrjPath} | grep ${Version1} | \
          grep --count -e '.orig.tar.[gx]z -> ')
        if [ $OrigLinkNr -ge 1 ]
        then
          OrigLink=$(ls -la ${PrjPath} | grep --regexp='.orig.tar.[gx]z -> ')
          OrigLink=$(echo ${OrigLink} | \
            sed --expression='s/^.*:.. //' | sed --expression='s/ //g')
          echo "${OrigLink}" will be transformed into a file" >> ${log}

          LinkTarget=$(echo $OrigLink | sed --expression='s/^.*->///')
          LinkName=$(echo $OrigLink | sed --expression='s/->.*$//')
          rm ${PrjPath}/${LinkName}
          cp -a ${PrjPath}/${LinkTarget} ${PrjPath}/${LinkName}
          whiptail --title "Result of transformation link to file:" \
            --msgbox "$(ls -la ${PrjPath})" --scrolltext 15 60
          echo "Result of transformation link to file: \
            "$(ls -la ${PrjPath}) >> ${log}
        fi
        set -e
      }

      <CheckGitStatus 170>

```

34.3.9. gbp-Konfigurationsdatei

gbp import-orig (Kapitel 34.3.11, Seite 218) fügt den heruntergeladenen Quellcode dem Git-Repositorium hinzu.

Zur Steuerung dieses Prozesses wird die Einfügung einer vorbereiteten Datei *gbp.conf* in das Verzeichnis *.git* ermöglicht (Kapitel 34.3.11, Seite 218)

```
210a <SearchGbpConf 210a>≡ (215b)
function SearchGbpConf {
    # Called by BuildNewVersion BuildWithUscan

    # Neither .git/gbp.conf nor debian/gbp.conf exist
    if [ ! -f ${GitPath}/.git/gbp.conf -a ! -f ${GitPath}/debian/gbp.conf ]
    then
        if whiptail --title "gbp.conf needed?" \
            --yesno "Do you want to integrate a special gbp.conf for this project?" \
            --defaultno --yes-button "Yes" --no-button "No" 15 60
        <SearchGbpConf1 210b>
```

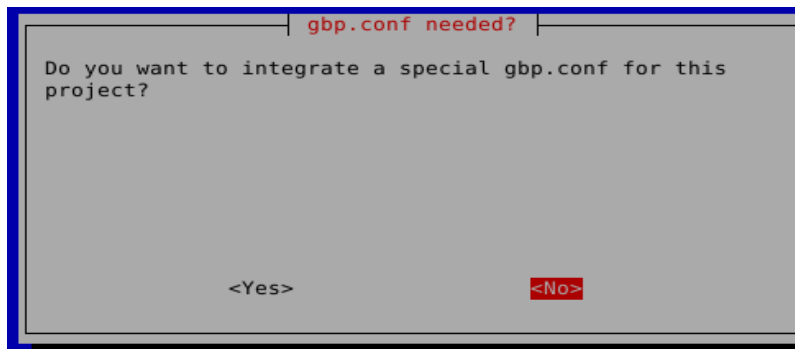


Abbildung 34.32.: Spezielle gbp.conf

Wird diese Frage verneint, geht es mit dem Import in das Git-Repositorium (Kapitel 34.3.11, Seite 218) weiter.

```
210b <SearchGbpConf1 210b>≡ (210a)
    then
        GbpConfIntegration
    fi
fi
<SearchGbpConf2 212a>
```

```
210c <GbpConfIntegration 210c>≡ (194a)
function GbpConfIntegration {
    # Called by SearchGbpConf and itself
    GbpConfPath=$(whiptail --title "gbp.conf" \
        --inputbox "Please insert the path to the your special\n \
        gbp.conf for this project:" \
        --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
    <GbpConfIntegration1 211>
```

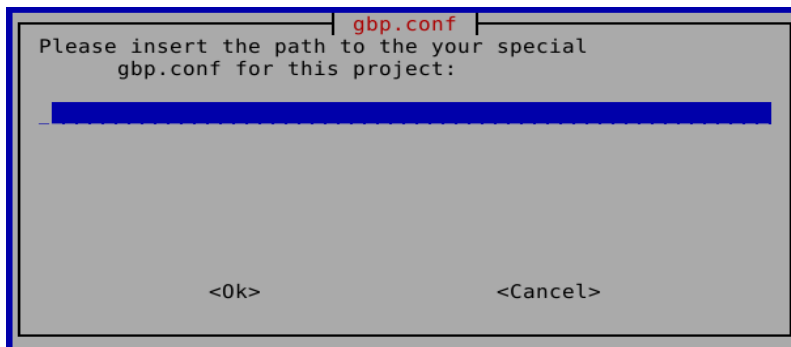


Abbildung 34.33.: Abfrage: Pfad zur gbp.conf

```

211  <GbpConfIntegration1 211>≡ (210c)
      if [ -z "${GbpConfPath}" ]
      then
          echo "Please insert the path to the your special gbp.conf for this project:"
          read GbpConfPath
      fi

      # Replace tilde if necessary
      SuspectPath=${GbpConfPath}
      ReplaceTilde
      GbpConfPath=${CleanPath}

      if [ -f ${GbpConfPath}/gbp.conf ]
      then
          cp -av ${GbpConfPath}/gbp.conf ${GitPath}/.git/
      else
          if whiptail --title "File not found!" \
          --yesno "There was no gbp.conf found at ${GbpConfPath}! Retry?" \
          --yes-button "Yes" --no-button "No" 15 60
          then
              GbpConfIntegration
          fi
      fi
  }

  <TwoConfFilesFound 214a>

```

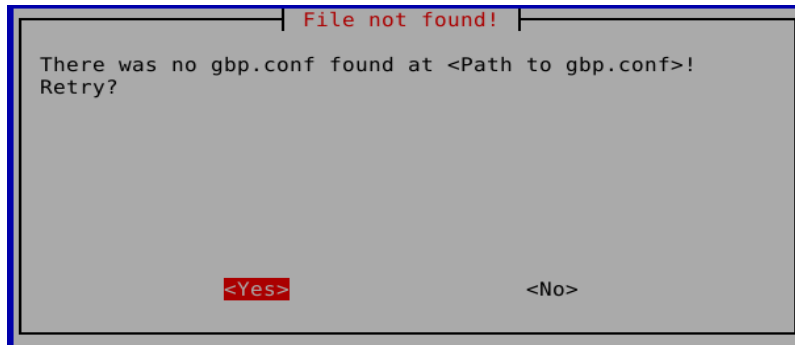


Abbildung 34.34.: gbp.conf nicht gefunden

Wird eine Datei *gbp.conf* gefunden, wird diese zwecks Überprüfung und gegebenenfalls Anpassung im Editor angezeigt. Zu prüfen ist besonders der Wert der Variablen *compression* – vor allem dann, wenn ein entsprechender Wechsel des Archivformats erfolgen soll.

```
212a <SearchGbpConf2 212a>≡ (210b)
      # debian/gbp.conf exists, but not .git/gbp.conf
      if [ ! -f ${GitPath}/.git/gbp.conf -a -f ${GitPath}/debian/gbp.conf ]
      then
          whiptail --title "Found gbp.conf" \
          --msgbox "Please check and edit your gbp.conf (if necessary)" 15 60
```

<SearchGbpConf3 212b>



Abbildung 34.35.: Check gbp.conf

```
212b <SearchGbpConf3 212b>≡ (212a)

      nano --linenumbers --mouse --softwrap ${GitPath}/debian/gbp.conf
      fi
      # .git/gbp.conf exists, but not debian/gbp.conf
      if [ -f ${GitPath}/.git/gbp.conf -a ! -f ${GitPath}/debian/gbp.conf ]
      then
          whiptail --title "Found gbp.conf" \
          --msgbox "Please check and edit your gbp.conf (if necessary)" 15 60
```

<SearchGbpConf4 213a>



Abbildung 34.36.: Check gbp.conf

213a `<SearchGbpConf4 213a>≡`

(212b)

```

    nano --linenumbers --mouse --softwrap ${GitPath}/.git/gbp.conf
  fi
  # There is a gbp.conf in both directories
  if [ -f ${GitPath}/.git/gbp.conf -a -f ${GitPath}/debian/gbp.conf ]
  then
    TwoConfFilesFound
  fi
}

```

`<MovingGbpConfFile 301b>`

Im folgenden wird die Datei mit den Informationen abgebildet, die für viele Debian-Pakete verwendet werden können.

213b `<debian/gbp.conf 213b>≡`

```

# Configuration file for git-buildpackage and friends

[DEFAULT]
# use pristine-tar:
pristine-tar = True
# generate xz compressed orig file
compression = xz
debian-branch = debian/experimental
upstream-branch = upstream

[pq]
patch-numbers = False

[dch]
id-length = 7
debian-branch = debian/experimental

[import-orig]
# filter out unwanted files/dirs from upstream
filter = [ '.cvsignore', '.gitignore', '.hgtags', '.hgignore', '*.orig', *.rej' ]
# filter the files out of the tarball passed to pristine-tar
filter-pristine-tar = True

```

Nach der Überprüfung und eventuellen Verbesserung der Datei *gbp.conf* geht es mit weiteren Vorbereitungen für den Import ins lokale Git-Repository weiter (Kapitel 34.3.10, Seite 216). Verläuft die Überprüfung des Git-Repositories befundlos, geht es direkt mit dem Import ins lokale Git-Repository weiter (Kapitel 34.3.11, Seite 218).

Die Funktion *TwoConfFilesFound* behandelt den Fall, dass jeweils eine Datei *gbp.conf* sowohl im Verzeichnis *.git/* als auch im Verzeichnis *debian/* vorkommt.

```
214a <TwoConfFilesFound 214a>≡ (211)
function TwoConfFilesFound {
    # Called by SearchGbpConf MovingGbpConfFile

    whiptail --title "Information" \
        --msgbox "There are a gbp.conf in debian/ and a gbp.conf in .git/" 15 60
<TwoConfFilesFound0-1 214b>
```

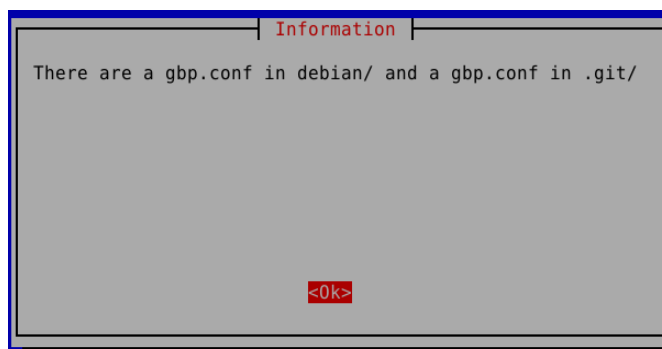


Abbildung 34.37.: *gbp.conf* zweimal gefunden.

```
214b <TwoConfFilesFound0-1 214b>≡ (214a)
# Are they different?
GitConfFile=$(cat ${GitPath}/.git/gbp.conf)
DebianConfFile=$(cat ${GitPath}/debian/gbp.conf)
if [ "${GitConfFile}" != "${DebianConfFile}" ]
then
    whiptail --title "Warning!" \
        --msgbox "There are a gbp.conf in debian/ and a gbp.conf in .git/\n \
        But they are different!\n\nThe left column is ${GitPath}/.git/gbp.conf\n \
        The right column is ${GitPath}/debian/gbp.conf\n \
        After studying the diff press RETURN!" 15 60

<TwoConfFilesFound1 215a>
```

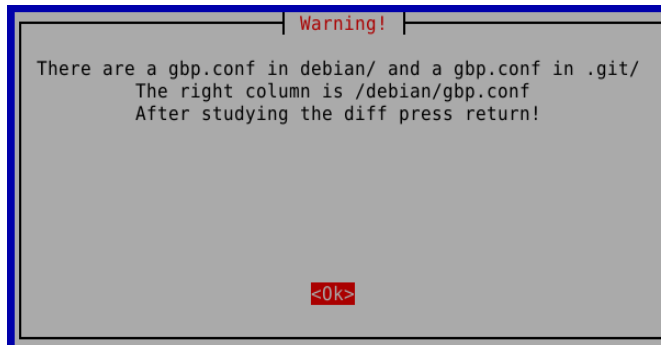


Abbildung 34.38.: Unterschiedliche Konfigurationsdateien

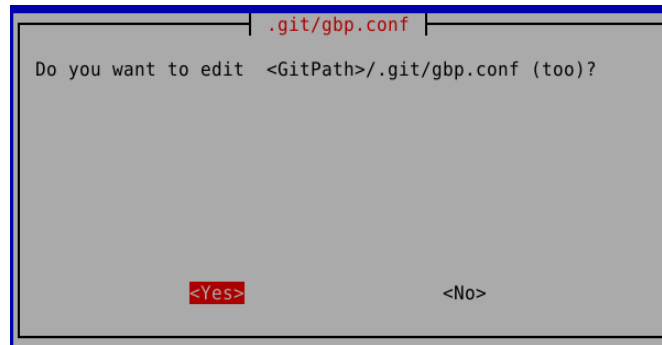
Mit `diff --side-by-side (-y)` wird die Differenz in zwei Spalten angezeigt.

```
215a  <TwoConfFilesFound1 215a>≡ (214b)
      diff --side-by-side ${GitPath}/.git/gbp.conf ${GitPath}/debian/gbp.conf
      read a
      fi
      # Editing
      if whiptail --title "debian/gbp.conf" \
        --yesno "Do you want to edit ${GitPath}/debian/gbp.conf?" \
        --yes-button "Yes" --no-button "No" 15 60
    <TwoConfFilesFound2 215b>
```

Abbildung 34.39.: `gbp.conf` im Verzeichnis `debian/` bearbeiten?

```
215b  <TwoConfFilesFound2 215b>≡ (215a)
      then
        nano --linenumbers --mouse --softwrap ${GitPath}/debian/gbp.conf
      fi
      if whiptail --title ".git/gbp.conf" \
        --yesno "Do you want to edit ${GitPath}/.git/gbp.conf (too)?" \
        --yes-button "Yes" --no-button "No" 15 60
      then
        nano --linenumbers --mouse --softwrap ${GitPath}/.git/gbp.conf
      fi
    }

    <SearchGbpConf 210a>
```

Abbildung 34.40.: *gbp.conf* im Verzeichnis *.git/* bearbeiten?

34.3.10. Prüfung des Git-Repositorys

Zur weiteren Vorbereitung des Importes in das lokale Git-Repositorys finden noch Prüfungen desselben statt.

```
216a <Import2Git 216a>≡ (17b)
function Import2Git {
    # Called by BuildNewVersion and itself

    CheckGitStatus # to exercise caution
    CheckTags
```

<Import2Git1 218b>

Es wird geprüft, ob noch eine lokale Änderung nicht zum Commit vorgemerkt wurde. Dies erfolgt mit der Funktion *CheckGitStatus* (Kapitel 33.4.1, Seite 170). Solche Änderungen müssen vor dem Import einer neuen Version abgearbeitet werden.

Eine weitere Prüfung erfolgt mit der Funktion *CheckTags*. Diese listet die vorhandenen *Tags* auf. Für den nachfolgenden Commit mit *gbp import-orig* darf der Tag der zu importierenden Version noch nicht existieren (Tag-Kollision).

Bleiben die Überprüfungen befundlos, erfolgt der Import der neuen Version in das Git-Repository (Kapitel 34.3.11, Seite 218).

```
216b <CheckTags 216b>≡ (170)
function CheckTags {
    # Called by BuildWithUscan Import2Git and itself
    # checks git tags before executing gbp import-orig
    echo $(git tag) >> ${log}
    set +e
    cTags=$(git tag | grep --fixed-strings ${Version1})
    set -e
    if [ ${#cTags} -gt 0 ]
    then
        cTags1=$(echo ${cTags} | sed --expression='s/ /\n/g')
        if ! whiptail --title "List of dubious tags:" \
            --yesno "${cTags1}\n\nDo you want to continue regardless?" --defaultno \
            --yes-button "Yes" --no-button "No" 15 60
        <CheckTags2 217a>
```

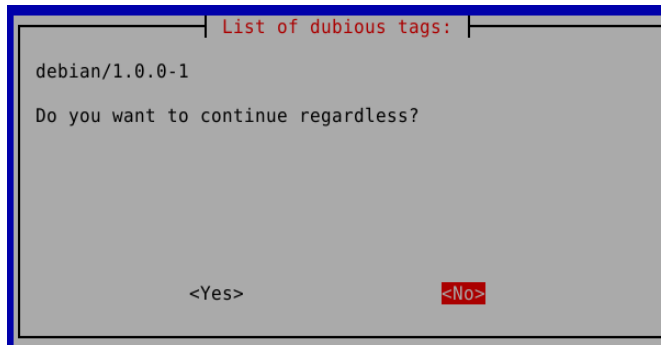


Abbildung 34.41.: Zweifelhafter Git-Tag

Wird hier ein `Git`-Tag aufgeführt, der vor dem Import einer neuen Version noch entfernt werden soll, lautet hier die Antwort „No“. Dann erfolgt im nächsten Dialog die Aufforderung, diesen `Git`-Tag zu entfernen.

```
217a  <CheckTags2 217a>≡ (216b)
      then
        whiptail --title "Delete tags!" \
          --msgbox "Please delete tags in another terminal\n \
          and then press ok" 15 60
      <CheckTags3 217b>
```



Abbildung 34.42.: Git Tags entfernen

```
217b  <CheckTags3 217b>≡ (217a)
      # git tag -d
      CheckTags
    fi
  fi
}
<Import2Git 216a>
```

34.3.11. Import nach Git

Hier wird zum ersten Mal *gbp* eingesetzt. Als erster Schritt wird mit *gbp import-orig* der heruntergeladene Quellcode dem Git-Repository hinzugefügt. Der Option *-debian-branch* wird der Inhalt der Variable *RecentBranch* zugewiesen.

Existieren mehrere mögliche Branches, kann der Debian-Branch, in den importiert werden soll, zuvor ausgewählt werden (Kapitel 33.4, Seite 169).

```
218a <BuildNewVersion15 218a>≡ (206b)
      Import2Git # Contains import to the git repo using gbp import
      Task=3 # Go to BuildNewRevision
    }
```

<DebianFormatTemplate 231b>

Die Funktion *Import2Git* (s. Kapitel 34.3.10, Seite 216) wird von der Funktion *BuildNewVersion* aufgerufen. Nach der Überprüfung des Git-Repositories wird der Zweig ermittelt, in den importiert werden soll.

```
218b <Import2Git1 218b>≡ (216a)
      # Check branch for import
      bl=$(git branch --list | sed --expression='s/* /x_/')
      ba=( $bl )
      set +e
      for element in ${ba[*]}
      do
          if echo ${element} | grep --quiet '^x_'
          then
              ActiveBranch=$(echo ${element} | sed --expression 's/\x_//')
          fi
      done
      set -e

      # After git init $ActiveBranch is always empty
      if [ ${ActiveBranch} ] && [ "${ActiveBranch}" != "${RecentBranch}" ]
      then
          whiptail --title "Check Branch!" \
          --msgbox "The active branch is ${ActiveBranch}.\n\
          In ${ConfigPath}${OrigName} 'RecentBranch' is ${RecentBranch}." \
          15 60
          echo -e "The active branch is "${ActiveBranch}".$\n\
          "In "${ConfigPath}${OrigName}" 'RecentBranch' is "${RecentBranch}".$\n\
          FailureNotice
      fi

      # Import to the git repo using gbp import

      echo "Notice from BuildNewVersion: The branch is \
      ${RecentBranch}" >> ${log}
      whiptail --title "Notice from BuildNewVersion:" \
      --msgbox "The branch is ${RecentBranch}" 15 60
    <Import2Git2 219a>
```

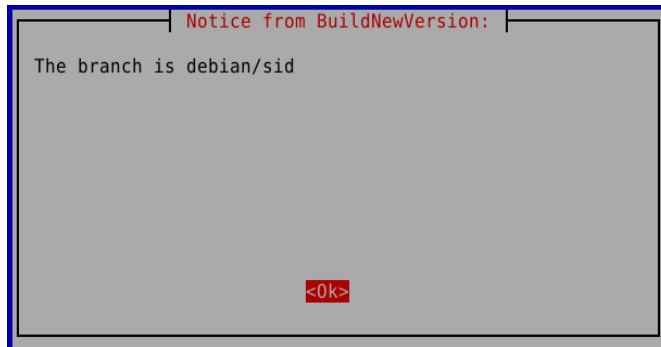


Abbildung 34.43.: Der Zweig is ...

```
219a <Import2Git2 219a>≡ (218b)
# The asterix (*) is for .bz2 and for lzma
OrigFile=$(ls ${PrjPath}/${SourceName}_${Version1}.orig.tar.*z*)
whiptail --title "Notice from BuildNewVersion:" \
--msgbox "The orig file to be imported is ${OrigFile}" 15 60
<Import2Git3 219b>
```

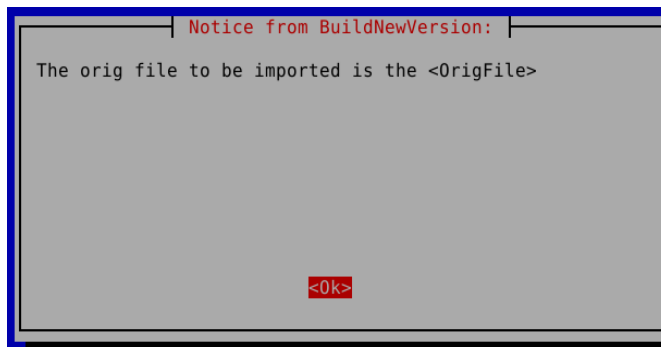


Abbildung 34.44.: Importiert wurde der Upstream-Code

Hier wird nun die Eingabe der Passphrase für den GPG-Schlüssel angefordert. Wird er nicht zeitnah eingegeben, wird der Import zurückgerollt. Es wird daher zunächst abgefragt, ob der GnuPG-Schlüssel zur Verfügung steht (Kapitel 32.8, Seite 160). Wird die Frage verneint, wird das Programm beendet.

Damit wird der zu erzeugende Git-Tag signiert. Dies entspricht guter Praxis. Nach *salsa.debian.org* sollten nur signierte Tags hochgeladen werden.

Das Signieren der Tags ist das Standardverhalten von *gbp import-orig*. Diese Option wurde gleichwohl in die Befehlszeile aufgenommen, denn „explizit ist besser als implizit“.

Will man ausnahmsweise mal nicht signieren, ist die Option *-no-sign-tags* zu verwenden.

```
219b <Import2Git3 219b>≡ (219a)

GpgKeyAvailable
echo -e "\n Starting gbp import-orig - Please wait"'\n"
# Signing tags is default
gbp import-orig --verbose --debian-branch=${RecentBranch} \
--sign-tags ${OrigFile}
```

```
<Import2Git4 220>
```

4. Mai 2025

Nun zeigt *gbp buildpackage* ausführlich alle Schritte, die nun ausgeführt werden. Dabei muss die ermittelte Paket-Version bestätigt oder angepasst werden.

```
220 <Import2Git4 220>≡ (219b)
      if [ $? -eq 0 ]
      then
        echo "${OrigFile}" was imported by gbp import-orig" >> ${log}
      else
        whiptail --title "Something went wrong!" \
          --msgbox "gbp import-orig -v --debian-branch=${RecentBranch} \
            --sign-tags ${OrigFile} failed!" 15 60
        echo "gbp import-orig -v --debian-branch=${RecentBranch}" \
          --sign-tags "${OrigFile}" failed!" >> ${log}
        FailureNotice
        Import2Git
      fi
}
```

<BuildNewVersion 186>

Am Ende ruft die Funktion *BuildNewVersion* die Funktion *BuildNewRevision* auf (Kapitel 35, Seite 225)

34.4. Herunterladen und Importieren mit *uscan*

Die folgenden Schritte werden durch den Befehl *gbp import-orig --uscan ...* ausgeführt.

Anhand des ersten Eintrages in der Datei *debian/changelog* (Kapitel 37.1, Seite 291) ermittelt *uscan* die Versionsbezeichnung des zuletzt gebauten Paketes. *uscan* lädt dann die Web-Seite von der in der Datei *debian/watch* (Kapitel 35.4.7, Seite 239) angegebenen *URL*. Dann sucht *uscan* unter Verwendung des in *debian/watch* angegebenen Suchmusters nach Hyperlinks (*href*), die auf Upstream-Archive verweisen.

uscan lädt das Upstream-Archiv mit der neuesten Version herunter, wenn diese neuer als die in *debian/changelog* zuletzt angegebene Version ist. Das heruntergeladene Archiv wird im übergeordnete Verzeichnis gespeichert. Schließlich wird *mk-origtar.gz* (s. Kapitel 19.1.1, Seite 57) aufgerufen.

Im Programmskript wird zunächst geprüft, ob ein Herunterladen mit *uscan* möglich und sinnvoll ist.

Ferner wird eine vorhandene Datei *gbp.conf* (mit der Funktion *SearchGbpConf* (Kapitel 34.3.9, Seite 210)) zum Zwecke der Prüfung im Editor geöffnet. Es ist besonders auf die angegebene Kompression (*compression*) zu achten.

```
221 <BuildWithUscan 221>≡ (223a)
function BuildWithUscan {
    # Called by ClassicalOrUscan

    cd ${GitPath}

    echo "Try gbp import-orig --uscan" >> ${log}
    set +e
    uscaninfo=$(uscan --no-download --verbose)
    if [ $#uscaninfo -gt 0 ]
    then
        SearchGbpConf
        whiptail --title "uscan" --msgbox "${uscaninfo}" \
            --scrolltext 15 60
        echo -e "Result of uscan:\n"${uscaninfo} >> ${log}
        set +e
        echo "${uscaninfo} | grep '=> Package is up to date' > /dev/null
        if [ $? -eq 0 ]
        then
            whiptail --title "uscan" \
                --msgbox "Package seems to be up to date.\n \
                Nothing to do!" 15 60
            echo "Package seems to be up to date. Nothing to do!" \
                >> ${log}
        <BuildWithUscan4 222a>
```

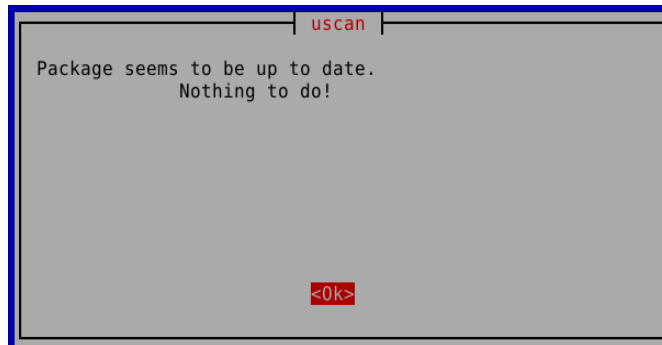


Abbildung 34.45.: Up to date

```
222a <BuildWithUscan4 222a>≡ (221)
fi
echo ${uscaninfo} | grep '=> Newer package available from' \
> /dev/null
if [ $? -eq 0 ]
then
    if ! whiptail --title "Newer package available" \
    --yesno "All well? Continue?" --yes-button "Yes" \
    --no-button "Exit" 15 60
    then
        exit
    fi
fi
<BuildWithUscan5 222b>
```

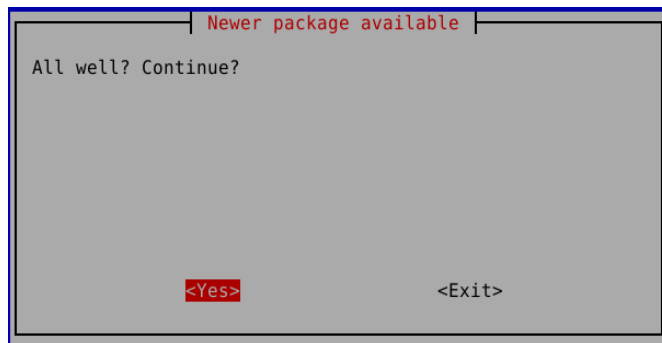


Abbildung 34.46.: Neue Version verfügbar

Es wird nun die Funktion *CheckRepackSuffix* ausgeführt.

```
222b <BuildWithUscan5 222b>≡ (222a)
set -e
CheckRepackSuffix
set +e
<BuildWithUscan5-1 223b>
```

Hier wird geprüft, ob in der Konfigurationsdatei der Eintrag *RecentRepackSuffix* vorhanden ist, aber kein entsprechender Eintrag in *debian/watch*.

In diesem Fall erfolgt ein Hinweis und die Datei *debian/watch* wird zum Editieren präsentiert.

```
223a  <CheckRepackSuffix 223a>≡ (328)
      function CheckRepackSuffix {
        # Called by BuildWithUscan
        if [ -n "${RecentRepackSuffix}" ]
        then
          if ! cat ${GitPath}/debian/watch | grep "repacksuffix=" > /dev/null
          then
            whiptail --title "debian/watch!" \
              --msgbox "No repacksuffix in debian/watch." 15 60
            nano --linenumbers --mouse --softwrap ${GitPath}/debian/watch
          fi
        fi
      }

      <BuildWithUscan 221>
```



Abbildung 34.47.: Kein Repack Suffix in debian/watch

Die Versionsbezeichnung der neuen Version wird durch einen Testlauf von *uscan* ermittelt.

Sodann wird die Funktion *CheckGitStatus* aufgerufen (Kapitel 33.4.1, Seite 170), um sicherzustellen, dass alle lokalen Änderungen committet wurden.

```
223b  <BuildWithUscan5-1 223b>≡ (222b)
      Version1=$(uscan --no-download --verbose | \
        grep newversion | sed --expression 's/  $newversion = //' )
      set -e
      CheckGitStatus
      <BuildWithUscan6 223c>
```

Zum Signieren des Archivs des heruntergeladenen Quellcodes wird der GPG-Schlüssel des Maintainers benötigt. Dieser muss daher zur Verfügung stehen. Sonst kann es nicht weitergehen. (Kapitel 32.8, Seite 160)

```
223c  <BuildWithUscan6 223c>≡ (223b)
      GpgKeyAvailable
      <BuildWithUscan7 224>
```

4. Mai 2025

Die eingangs beschriebenen Schritte werden durch `gbp import-orig --uscan` ausgeführt. Für Einzelheiten kann auf die Handbuchseite von `gbp import-orig` [41] verwiesen werden.

```
224 <BuildWithUscan? 224>≡ (223c)
    CheckTags
    set +e
    # Downloads with uscan and imports with gbp import-orig
    gbp import-orig --uscan --verbose \
    --debian-branch=${RecentBranch} --sign-tags ${OrigFile}
    if [ $? -ne 0 ]
    then
        echo "Import with gbp import-orig --uscan failed!" \
        >> ${log}
        exit
    fi
    echo "Imported with gbp import-orig --uscan" >> ${log}
else
    exit
fi
set -e
else
    whiptail --title "Uscan failed!" \
    --msgbox "Please check the watch file with uscan." 15 60
    echo "Uscan failed! Please check the watch file with uscan." >> ${log}
    exit
fi

Task=3 # Go to BuildNewRevision
}
```

<PrepareUploading (nicht definiert)>

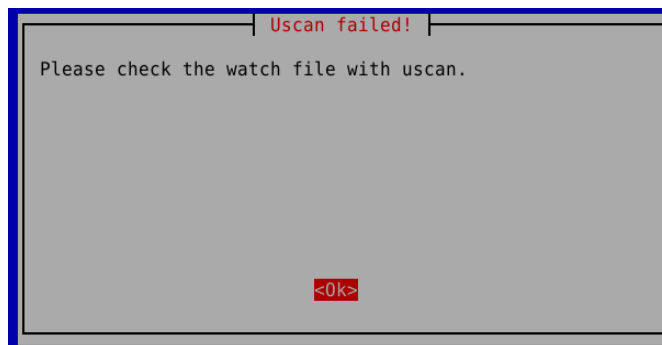


Abbildung 34.48.: Uscan kann Watch-Datei nicht parsen

Am Ende ruft die Funktion `BuildWithUscan` die Funktion `BuildNewRevision` auf.

35. Bauen einer neuen Revision

Die Funktion *BuildNewRevision* wird automatisch von den Funktionen *BuildNewVersion* und *BuildWithUscan* aufgerufen.

Um das Bauen einer neuen Revision auf einen späteren Zeitpunkt verschieben zu können, wird die Möglichkeit gegeben, das Programm zuvor abzubrechen.

```
225a <BuildNewRevision2 225a>≡ (226)
    # Intro
    if ! whiptail --title "New Debian revision" \
        --yesno "A new Debian revision will be built." --yes-button "Yes" \
        --no-button "Exit" 15 60
    <BuildNewRevision3 225b>
```

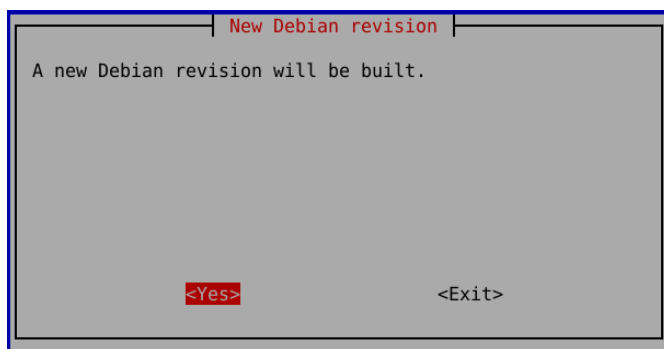


Abbildung 35.1.: Neue Revision bauen

```
225b <BuildNewRevision3 225b>≡ (225a)
    then
        exit
    fi

    echo "A new Debian revision will be built." >> ${log}
    <BuildNewRevision4 227>
```

Existiert das Verzeichnis *debian/source* und handelt es sich nicht um ein **Maven**-Paket geht es mit der Frage weiter, ob die Dateien im Verzeichnis *debian/* zum Editieren angezeigt werden sollen (Kapitel 35.3, Seite 228)

35.1. Anlegen des Debian-Verzeichnisses

Sofern das Verzeichnis *debian/source* (noch) nicht existiert, wird es vom Programmskript angelegt. Damit wird zugleich sichergestellt, dass auch das Verzeichnis *debian/* existiert (Kapitel 35.4, Seite 229).

Dies ist in der Regel nur dann relevant, wenn ein neues Paket für **Debian** gepackt werden soll.

```
226 <BuildNewRevision 226>≡ (308a)
    function BuildNewRevision {
        # Called by TaskSelect
        cd ${GitPath}

        ## Generate directory if necessary
        echo $(pwd) >> ${log}
        if [ -d debian/source ]
        then
            echo "The directory debian/source in ${GitPath} \
            already exists." >> ${log}
            dfe=1
        else
            mkdir --parents debian/source
            echo "Directory debian/source was created" >> ${log}
            dfe=0
        fi
    }

<BuildNewRevision2 225a>
```

Das Ergebnis der Ausführung des Programmskriptes wird in der Log-Datei vermerkt.

35.2. Abfrage: Bauen mit *mh-make*?

Sofern das Maven-Plugin (Kapitel 47, Seite 385) installiert ist und Maven als Build-System ausgewählt wurde, wird gefragt, ob bestimmte Dateien für dieses Build-System erstellt werden sollen. In der Regel kann diese Frage verneint werden.

227 `<BuildNewRevision4 227>`≡ (225b)

```
# For building java packages with maven

# To avoid an error, if 'MavenPluginFlag' is empty
if [ ! -z ${MavenPluginFlag} ] && [ ${MavenPluginFlag} -eq 1 ]
then
  . build-gbp-maven-plugin
  if whiptail --title "Maven" \
    --yesno "Should mh_make create the ${PackName}.poms\n \
    file and some maven.* files?\n\n \
    Normally you only need it at the first run" --yes-button "Yes" \
    --no-button "No" --defaultno 15 60
  then
    MakeMaven
  fi
fi
```

`<BuildNewRevision5 228>`

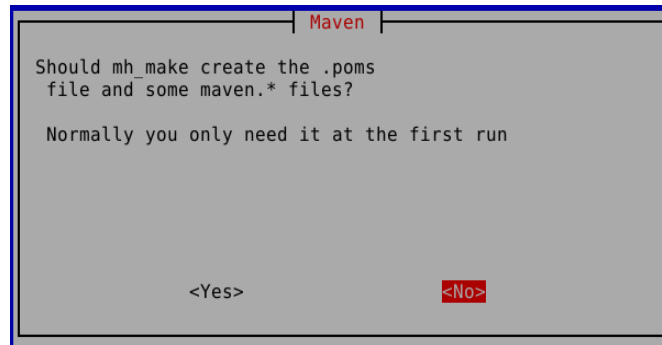


Abbildung 35.2.: Daten für Maven erstellen?

Wird sie bejaht, wird das **Maven**-Plugin geladen (Kapitel 47, Seite 385) und die Funktion *MakeMaven* aufgerufen. In Kapitel 47.3 (Seite 386) werden die weiteren Schritte beschrieben.

35.3. Sollen die Debian-Dateien angezeigt werden?

Hier wird nun die Möglichkeit eröffnet, die Dateien im Verzeichnis *debian/* zu erstellen und zu editieren.

```
228 <BuildNewRevision5 228>≡ (227)
    # Displaying files in debian/ for editing
    if [ ${dfe} -ne 1 ]
    then
        DisplayDebianFiles
    else
        if whiptail --title "Showing debian files for editing" \
        --yesno "Should the files of debian/ be displayed\n \
        to check, edit or create them?" --yes-button "Yes" \
        --no-button "No" 15 60
        then
            DisplayDebianFiles
        fi
    fi
    dfe=0
```

<BuildNewRevision5-1 253a>

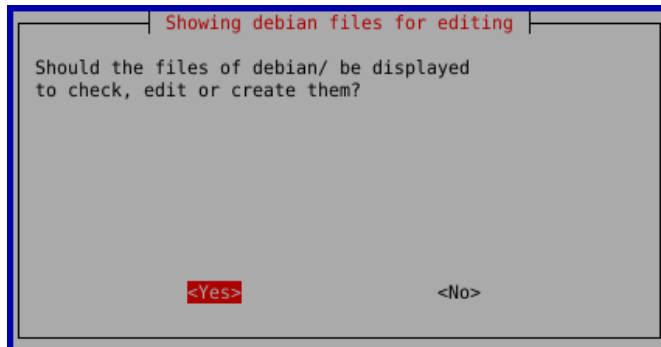


Abbildung 35.3.: Debian-Dateien anzeigen

Wird die Frage, ob die Dateien im Verzeichnis *debian/* angezeigt werden sollen, bejaht, werden diese Dateien, soweit notwendig, erstellt und zum möglichen Editieren angezeigt.

Andernfalls geht es mit Änderungen am Upstream-Code weiter (Kapitel 36, Seite 253) und es wird lediglich noch die Datei *debian/changelog* angezeigt (Kapitel 37.1, Seite 291).

35.4. Dateien im Verzeichnis *debian/*

Die Dateien im Verzeichnis *debian/* dienen der Steuerung und der Dokumentation des Build-Prozesses. Die Dateien in diesem Verzeichnis werden in der Archiv-Datei mit der Endung *<Paketname>.debian.tar.xz* veröffentlicht (s. Kapitel 8, Seite 23).

Das Verzeichnis *debian/* wird, wenn nötig, vom Programmskript erstellt (Kapitel 35.1, Seite 226).

Das Skript hat die Aufgabe, die notwendigen oder häufiger verwendeten Dateien im Verzeichnis *debian/* zu erstellen und - soweit möglich - Vorschläge für ihren Inhalt zu machen.

Existieren die Debian-Dateien bereits, kann der Nutzer sie prüfen und verbessern.

35.4.1. Anzeigen der Debian-Dateien

Sollen die Dateien im Verzeichnis *debian/* angezeigt werden, um sie zu prüfen, zu editieren oder auch zu erzeugen, wird die folgende Funktion ausgeführt:

```
229 <DisplayDebianFiles 229>≡ (252)
    function DisplayDebianFiles {
        # Called by BuildNewRevision

        # Add Debian files

        # If the Debian files already exists, you can review and improve them now
        # If not, you have to write them

        ## There is default content for Debian files
        ## Change the default values, if you know, what you are doing

    <DisplayDebianFiles0 230a>
```

Ist in der Konfigurationsdatei ein entsprechender Eintrag vorhanden, wird das Plugin-Script für *Webext*-Pakete geladen (Kapitel 48, Seite 399)

Das Plugin-Skript zeigt an, dass es geladen wurde (s. Kapitel 48.2.7, Seite 403).

```
230a <DisplayDebianFiles0 230a>≡ (229)
      # Loading Webext plugin or Python3 Plugin if needed

      if [ ${WebextFlag} -eq 1 ]
      then
        . build-gbp-webext-plugin
      fi
```

<DisplayDebianFiles01 230b>

Ist in der Konfigurationsdatei ein entsprechender Eintrag vorhanden, wird das Plugin-Script für *Python*-Pakete geladen (Kapitel 48, Seite 399)

Das Plugin-Skript zeigt an, dass es geladen wurde (s. Kapitel 49.0.1, Seite 405).

```
230b <DisplayDebianFiles01 230b>≡ (230a)
      if [ ${PythonFlag} -eq 1 ]
      then
        . build-gbp-python-plugin
      fi
```

<DisplayDebianFiles02 230c>

Es werden dann Funktionen aufgerufen, die gegebenenfalls Vorlagen für Dateien im Verzeichnis *debian/* erstellen.

```
230c <DisplayDebianFiles02 230c>≡ (230b)
```

```
      DebianFormatTemplate
      <DisplayDebianFiles1 230d>
```

Die Funktion *DebianFormatTemplate* erstellt eine Vorlage für die Datei *debian/source/format* (Kapitel 35.4.2, Seite 231).

```
230d <DisplayDebianFiles1 230d>≡ (230c)
      DebianUpstreamMetadataTemplate
      <DisplayDebianFiles2 230e>
```

Kapitel 35.4.4, Seite 233

```
230e <DisplayDebianFiles2 230e>≡ (230d)
      DebianCopyrightTemplate
      <DisplayDebianFiles3 230f>
```

```
230f <DisplayDebianFiles3 230f>≡ (230e)
      DebianControlTemplate
      <DisplayDebianFiles4 230g>
```

```
230g <DisplayDebianFiles4 230g>≡ (230f)
      DebianWatchTemplate
      <DisplayDebianFiles4-1 230h>
```

```
230h <DisplayDebianFiles4-1 230h>≡ (230g)
      DebianRulesTemplate
      <DisplayDebianFiles4-2 230i>
```

```
230i <DisplayDebianFiles4-2 230i>≡ (230h)
      DebianSalsaCiTemplate
```

<DisplayDebianFiles5 248a>

Wenn ein Java-Paket ohne Build-System gebaut werden soll, kann es notwendig sein, eine Datei *debian/javabuild* zu erstellen.

Wenn das Maven-Plugin verwendet wird, werden die Maven-Dateien zum Editieren angezeigt (Kapitel 47.4, Seite 394).

```
231a <DisplayDebianFiles10 231a>≡ (250)
    if [ ${JavaFlag} -eq 1 ]
    then
        if [ ${MavenPluginFlag} -eq 0 ]
        then
            DebianJavabuildTemplate
        else
            ls debian/ | grep 'maven'
            if [ $? -eq 0 ]
            then
                ShowMaven
            fi
        fi
    fi
    CmeFix
}

<ForceOrig 316b>
```

35.4.2. debian/source/format

Diese Datei enthält das Format des Quellpaketes. In der Datei *debian/source/format* wird der Eintrag *3.0 (quilt)* erstellt. Dies bedeutet, dass es sich um *kein* natives Paket handelt. Bei einem nativen Paket muss *3.0 (native)* eingetragen werden.

Was ein natives Paket ist, wird in Kapitel 16.1 (Seite 51) und in Kapitel 4 der *Debian-Policy* [7] beschrieben.

Eine ausführliche Beschreibung zur Datei *debian/source/format* gibt es im *Debian-Leitfaden für Neue Paketbetreuer*[11] in Kapitel 5.22 ¹

```
231b <DebianFormatTemplate 231b>≡ (218a)
function DebianFormatTemplate {
    # Called by DisplayDebianFiles

    if ! [ -f ${GitPath}/debian/source/format ]
    then
        # String for debian/source/format
        if whiptail --title "Kind of package" \
            --defaultno --yesno "Is it a native debian package?" \
            --yes-button "Yes" --no-button "No" 15 60

        <DebianFormatTemplate1 232>
```

¹<https://3ws.debian.org/doc/manuals/maint-guide/dother.de.html#sourcef>

4. Mai 2025

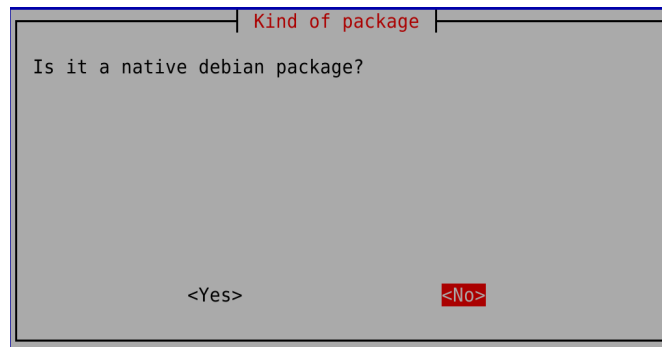


Abbildung 35.4.: Art des Quellpaketes (Nativ? J/N)

```
232  <DebianFormatTemplate1 232>≡ (231b)
      then
        str4format="3.0 (native)"
      else
        str4format="3.0 (quilt)"
      fi

      touch ${GitPath}/debian/source/format
      echo ${str4format} >> ${GitPath}/debian/source/format
      echo "/debian/source/format was created." >> ${log}
    fi
    nano --linenumbers --mouse --softwrap ${GitPath}/debian/source/format
  }

  <DebianUpstreamMetadataTemplate 233>
```

35.4.3. debian/source/include.binaries

Grundsätzlich sollen Binär-Dateien keine Aufnahme in das Debian-Paket finden. Sie sind daher regelmäßig von der Aufnahme in das *.orig-Archiv auszuschließen (Kapitel 34.3.5, Seite 197).

Es gibt jedoch Ausnahmen. Hierzu können Medien-Dateien und komprimierte Dokumentationen zählen. Diese Dateien sind zwecks Dokumentation in der Datei *debian/source/include-binaries* unter Angabe ihres Pfades aufzuführen.

35.4.4. debian/upstream/metadata

Es gibt eine ausführliche Beschreibung^[42] dieser *YAML*-Datei in englischer Sprache. Dort wird auch auf DEP-12² hingewiesen.

Dort wird auch erläutert, welche Informationen in den einzelnen Zeilen vom Maintainer eingetragen werden sollen, sofern diese Informationen vorhanden sind.

Es wird empfohlen, beim Editieren der Datei die Kommentare zu entfernen.

```
233 <DebianUpstreamMetadataTemplate 233>≡ (232)
function DebianUpstreamMetadataTemplate {
    # Called by DisplayDebianFiles

    # Treatment for native packages
    set +e
    cat debian/source/format | grep "native" > /dev/null
    if [ $? -eq 0 ]
    then
        set -e
        return
    fi
    set -e

    # Strings for debian/upstream/metadata
    if ! [ -f ${GitPath}/debian/upstream/metadata ]
    then
        mkdir --parents debian/upstream
        # creating a template for debian/upstream/metadata
        echo -e "# You can find a description at\n#\n\
https://wiki.debian.org/UpstreamMetadata" \
        >> debian/upstream/metadata
        echo "# Archive: " >> debian/upstream/metadata
        echo "# ASCL-id: " >> debian/upstream/metadata
        echo "Bug-Database: " >> debian/upstream/metadata
        echo "Bug-Submit: " >> debian/upstream/metadata
        echo "# Cite-As: " >> debian/upstream/metadata
        echo "Changelog: " >> debian/upstream/metadata
        echo "# CPE: " >> debian/upstream/metadata
        echo "Documentation: " >> debian/upstream/metadata
        echo "# Donation: " >> debian/upstream/metadata
        echo "# FAQ: " >> debian/upstream/metadata
        echo "# Funding: " >> debian/upstream/metadata
        echo "# Gallery: " >> debian/upstream/metadata
        echo "# Other-References: " >> debian/upstream/metadata
        echo "# Reference: " >> debian/upstream/metadata
        echo "#     Author: " >> debian/upstream/metadata
        echo "#     Booktitle: " >> debian/upstream/metadata
```

²Quelle:[43]

```

echo "#      DOI: " >> debian/upstream/metadata
echo "#      Editor: " >> debian/upstream/metadata
echo "#      Eprint: " >> debian/upstream/metadata
echo "#      ISBN: " >> debian/upstream/metadata
echo "#      ISSN: " >> debian/upstream/metadata
echo "#      Journal: " >> debian/upstream/metadata
echo "#      Number: " >> debian/upstream/metadata
echo "#      Pages: " >> debian/upstream/metadata
echo "#      PMID: " >> debian/upstream/metadata
echo "#      Publisher: " >> debian/upstream/metadata
echo "#      Title: " >> debian/upstream/metadata
echo "#      Type: " >> debian/upstream/metadata
echo "#      URL: " >> debian/upstream/metadata
echo "#      Volume: " >> debian/upstream/metadata
echo "#      Year: " >> debian/upstream/metadata
echo "#      Debian-package: " >> debian/upstream/metadata
echo "# Registration: " >> debian/upstream/metadata
echo "# Registry: " >> debian/upstream/metadata
echo "Repository: " >> debian/upstream/metadata
echo "Repository-Browse: " >> debian/upstream/metadata
echo "# Screenshots: " >> debian/upstream/metadata
echo "# Security-Contact: " >> debian/upstream/metadata
echo "# Webservice: " >> debian/upstream/metadata
fi
nano --linenumbers --mouse --softwrap ${GitPath}/debian/upstream/metadata
}

```

(DebianCopyrightTemplate 234)

Folgende Felder sind für viele Pakete relevant. Ihr Vorhandensein wird teilweise von *lintian* geprüft.

Bug-Database URL zur Liste der bekannten Fehler

Bug-Submit Adresse, an die Fehlermeldungen gesandt werden können.

Changelog URL des Upstream Changelogs

Documentation Upstream Dokumentation

Repository URL zum Upstream-Repositorium

Repository-Browse Durchsuchbares Repositorium von Upstream

35.4.5. debian/copyright

Diese Datei enthält Informationen über das Copyright und die Lizenzen der Quellen der Originalautoren.

Diese Datei kann mit *debmake -cc* erzeugt werden und im DEP-5-Format [19] abgelegt

```

234 (DebianCopyrightTemplate 234)≡ (233)
function DebianCopyrightTemplate {
    # Called by DisplayDebianFiles

    if ! [ -f ${GitPath}/debian/copyright ]
    then
        # creating debian/copyright using debmake
        debmake -cc > debian/copyright
    }
(DebianCopyrightTemplate2 235a)

```

Auszug aus der Manpage für *debmake*

- ```
-c, --copyright
 scan source for copyright+license text and exit.
```
- `-c`: simple output style
  - `-cc`: normal output style (similar to the `debian/copyright` file)
  - `-ccc`: debug output style

Danach muss die Datei noch bearbeitet werden. Dateien mit gleichem Autor und gleicher Lizenz können zusammengefasst werden. Stehen Dateien unter mehreren Lizenzen, werden diese Lizenzen mit *or* verbunden.

```
235a <DebianCopyrightTemplate2 235a>≡ (234)
 fi
 nano --linenumbers --mouse --softwrap ${GitPath}/debian/copyright
}
```

<TeamMaintainer 138>

Jeder Abschnitt *Files* in der maschinenlesbaren Copyright-Datei muss auf eine Lizenz verweisen, für die jeweils ein eigenständiger Lizenzabsatz existiert. Diese Absätze müssen **nach** allen *Files*-Absätzen erscheinen.

Eigenständige Lizenzabsätze können verwendet werden, um den vollständigen Lizenztext für eine bestimmte Lizenz nur einmal bereitzustellen, anstatt ihn in jedem *Files*-Abschnitt zu wiederholen, der auf sie verweist.[44]

Steht der Lizenztext unter `/usr/share/common-licenses/` zur Verfügung, ist statt des kompletten Lizenztextes eine Kurzfassung und der Dateiname nebst Pfad der Datei, die den Lizenztext enthält, (beispielsweise `/usr/share/common-licenses/GPL-3`) aufzuführen.

### 35.4.6. *debian/control*

Diese Datei enthält essentielle Werte, die durch die Paketverwaltungswerkzeuge verwendet werden.

Das Paketverwaltungssystem verarbeitet Daten, die in einem gemeinsamen Format, den sogenannten Kontrolldaten, in der *control*-Datei gespeichert ist. Diese Daten werden für Quellpakete, Binärpakete und die *.changes*-Dateien verwendet, die die Installation der hochgeladenen Dateien steuern.

Einzelheiten werden in der *Debian-Policy*[7] beschrieben.

#### 35.4.6.1. Grundlegender Aufbau

Das Programmskript erzeugt ein „Grundgerüst“ der *control*-Datei für das Quellpaket. Die *control*-Datei des Binärpaketes und der *.changes*-Datei werden durch den Build-Prozess aus den Informationen des Abschnittes für die Binär-Datei(en) erstellt.

```
235b <DebianControlTemplate 235b>≡ (144a)
 function DebianControlTemplate {
 # Called by DisplayDebianFiles

 # Strings for debian/control
 str4versiondebhelpers="(=13)"
 }
 <DebianControlTemplate1 236a>
```

Ab der Version *debhelper*  $\geq 12$  wird die *Kompatibilitatsversion* nicht mehr in der Datei *debian/compat* zusatzlich gepflegt. Stattdessen wird in der Datei *debian/control* der Eintrag *debhelper* durch *debhelper-compat* mit der Version (= 13) ersetzt.<sup>3</sup> Dies gilt auch fur alle darauf folgenden Versionen.

Von der Verwendung der Version 11 wird bereits abgeraten.

```
236a <DebianControlTemplate1 236a>≡ (235b)
 str4standardsversion="4.7.2"
```

```
<DebianControlTemplate2 236b>
```

In der Datei *debian/control* muss es einen Eintrag „Standard-Version: “ geben, der die Konformitat zur Version der Debian-Policy (s. Kapitel 7.2, Seite 21) angibt. Hier wird die jeweils aktuelle Version (*jetzt: 4.7.2*) vorgegeben.

An dieser Stelle im Programmskript wird eine Vorlage fur die Datei *debian/control* erstellt, wenn die Datei noch nicht existiert.

```
236b <DebianControlTemplate2 236b>≡ (236a)
 if ! [-f ${GitPath}/debian/control]
 then
 # creating a template for debian/control
 echo -e "Source: ${SourceName} > debian/control
 echo -e "Priority: optional" >> debian/control
```

```
<DebianControlTemplate3 237c>
```

Name und E-Mail-Adresse von **Maintainer** und gegebenenfalls **Uploaders** werden durch die Funktion *DEBValues* ermittelt. (Kapitel 32.4.2, Seite 136)

```
236c <DebianControlTemplate4 236c>≡ (237c)
 DEBValues

 if [-n "${Maintainer}"]
 then
 echo -e "Maintainer: ${Maintainer} >> debian/control
 else
 echo "Maintainer: " >> debian/control
 fi

 if [-n "${Uploaders}"]
 then
 echo -e "Uploaders: ${Uploaders} >> debian/control
 fi
```

```
<DebianControlTemplate5 236d>
```

```
236d <DebianControlTemplate5 236d>≡ (236c)
 echo -e "Build-Depends: debhelper-compat" \
 ${str4versiondebhelpers} >> debian/control
<DebianControlTemplate6 237d>
```

<sup>3</sup>[https://release.debian.org/bookworm/freeze\\_policy.html](https://release.debian.org/bookworm/freeze_policy.html) (2023)



```

237a <DebianControlTemplate7 237a>≡ (237d)
 echo -e "Standards-Version: "${str4standardsversion} \
 >> debian/control
 echo -e "Rules-Requires-Root: no" >> debian/control
 echo -e "Vcs-Git: https://salsa.debian.org/"${SalsaName} \
 >> debian/control
 BrowserName=$(echo ${SalsaName} | sed --expression='s/.git$/g')
 echo -e "Vcs-Browser: https://salsa.debian.org/"${BrowserName} \
 >> debian/control
 echo -e "Homepage: \n" >> debian/control
 <DebianControlTemplate8 237b>

```

Nun folgt in der Datei *debian/control* die Informationen über das Binär-Paket.

```

237b <DebianControlTemplate8 237b>≡ (237a)
 echo -e "Package: "${PackName} >> debian/control
 echo -e "Architecture: all" >> debian/control
 echo -e "Depends: \${misc:Depends}" >> debian/control
 echo -e "Description: " >> debian/control
 echo "A template for debian/control was created." >> ${log}
 <DebianControlTemplate9 238a>

```

### 35.4.6.2. Anpassungen für Java-Pakete

Informationen aus dieser Datei werden bei der Erstellung der *control*-Datei des Binär-Paketes verwendet.

Für das Paketieren von Java-Paketen können schon folgende Einträge vorgenommen werden.

```

237c <DebianControlTemplate3 237c>≡ (236b)
 if [${JavaFlag} -eq 1]
 then
 echo -e "Section: java" >> debian/control
 else
 echo -e "Section:" >> debian/control
 fi
 <DebianControlTemplate4 236c>

```

Bei Paketen, die im Team betreut werden, wird hier die Adresse des Teams angegeben. Dies ist in der Regel die E-Mail-Adresse der Mailingliste. In diesen Fällen ist auch das Feld *Uploaders* mit den Namen der Paketbetreuer zu füllen.

*Maintainer* für das Java-Team ist beispielsweise *Debian Java maintainers* <pkg-java-maintainers@lists.aliases.debian.org>.

Damit der Paketierer nicht immer seinen vollen Namen und seine E-Mail-Adresse schreiben muss, sucht das Skript diese Daten zunächst in der Konfigurationsdatei (Kapitel 32.4.2, Seite 136).

```

237d <DebianControlTemplate6 237d>≡ (236d)
 if [${JavaFlag} -eq 1]
 then
 echo " , default-jdk" >> debian/control

 if [${MavenPluginFlag} -eq 1]
 then
 echo " , maven-debian-helper" >> debian/control
 fi
 fi
 <DebianControlTemplate7 237a>

```

**35.4.6.3. Web-Extension-Plugin**

Aufruf der Funktion zur Anpassung der Datei *debian/control* für Mozilla-AddOns. Diese Funktion befindet sich im Webext-Plugin (Kapitel 48.2.3, Seite 402)

```
238a <DebianControlTemplate9 238a>≡ (237b)
 if [${WebextFlag} -eq 1]
 then
 WebextControl
 fi
 <DebianControlTemplate10 238b>
```

**35.4.6.4. Python-Plugin**

Aufruf der Funktion zur Anpassung der Datei *debian/control* für Python-Pakete und -Bibliotheken. Diese Funktion befindet sich im Python-Plugin (Kapitel 49.2, Seite 407)

```
238b <DebianControlTemplate10 238b>≡ (238a)
 if [${PythonFlag} -eq 1]
 then
 PythonControl
 fi
 <DebianControlTemplate11 238c>

238c <DebianControlTemplate11 238c>≡ (238b)
 fi
 nano --linenumbers --mouse --softwrap ${GitPath}/debian/control
 }

 <OptionsWatchFile 240a>
```

### 35.4.7. debian/watch

Die Datei *watch* im Debian-Verzeichnis enthält Daten für das Programm *uscan* (Kapitel 34.4, Seite 221 und Kapitel 40.4, Seite 332).

Zur Bestimmung des Namens des Quellcodepaketes liest *uscan* den ersten Eintrag in der Datei *debian/changelog* (Kapitel 37.1, Seite 291). Anhand dieses Eintrages ermittelt *uscan* auch die Versionsbezeichnung des zuletzt gebauten Paketes [45].

Dann verarbeitet *uscan* die Zeilen der Datei *debian/watch* in einem Zuge von oben nach unten. Einzelheiten zur Datei *debian/watch* werden im entsprechenden Artikel im Debian-Wiki beschrieben<sup>4</sup>.

In dieser Datei können reguläre Ausdrücke im Perl-Format<sup>5</sup> verwendet werden. Zeilen, die mit einem *#* beginnen, werden als Kommentarzeilen ignoriert.

Eingangs dieser Datei steht die Version des verwendeten Formats. Diese Angabe ist erforderlich. Die empfohlene Versionsnummer ist die *4* und wird vom Skript schon bei der Erstellung der Datei eingetragen.

239a `<DebianWatchTemplate 239a>≡` (242b)

```
function DebianWatchTemplate {
 # Called by DisplayDebianFiles

 # Treatment for native packages
 set +e
 cat debian/source/format | grep "native" > /dev/null
 if [$? -eq 0]
 then
 set -e
 return
 fi
 set -e
```

`<DebianWatchTemplate1 239b>`

In der ersten Zeile der Datei *debian/watch* steht die Version des Programmes *uscan*.

239b `<DebianWatchTemplate1 239b>≡` (239a)

```
String for debian/watch
str4watch="version=4"

if ! [-f ${GitPath}/debian/watch]
then
creating a template for debian/watch
echo ${str4watch} > debian/watch
OptionsWatchFile
```

`<DebianWatchTemplate3 243>`

<sup>4</sup><https://wiki.debian.org/debian/watch>

<sup>5</sup>siehe hierzu:

- [https://de.wikibooks.org/wiki/Perl-Programmierung:\\_Reguläre\\_Ausdrücke](https://de.wikibooks.org/wiki/Perl-Programmierung:_Reguläre_Ausdrücke)
- <http://www.mathe2.uni-bayreuth.de/perl/GK/regExp.htm>
- [http://perl-seiten.privat.t-online.de/html/perl\\_reg.html](http://perl-seiten.privat.t-online.de/html/perl_reg.html)

Nun werden die Optionen definiert, wie *uscan* überprüfen kann, ob auch die aktuelle Version gebaut wird.

Die Auswertung der Versionierung folgt der Darstellung in Kapitel 11 (Seite 35).

Die Optionen geben Regeln für die Auswahl möglicher Upstream-Archive vor. Sie werden in der Handbuchseite (Manpage) von *uscan* erläutert <sup>6</sup>.

Das Programmskript stellt die Optionen auf der Basis der bisher vorhandenen Informationen in die Datei *debian/watch* ein.

Das Programmskript vermeidet Leerzeichen in der Liste der Optionen. Andernfalls muss die Optionenliste von doppelten Anführungszeichen (") eingerahmt werden.

```
240a <OptionsWatchFile 240a>≡ (238c)
function OptionsWatchFile {
 # Called by DebianWatchfile

 o1f='\\n'
 WOpt='opts='

```

<OptionsWatchFile1 240b>

Bei der Erstellung der *\*.orig.tar.\**-Datei (Kapitel 34.3.1, Seite 186) wurde bereits das ursprüngliche Archiv-Format festgestellt. Diese Information wird nun für die Datei *debian/watch* verwendet.

Die folgende Option legt beim Bauen einer neuen Version mittels *uscan* fest, dass die *\*.orig.tar.\**-Datei in einem anderen Kompressionsformat archiviert wird, als das herunterzuladende Upstream-Archiv. Angegeben wird hier als Kompressionsformat des orig-Archives *xz*.

Wenn das Upstream-Archiv in einem Zip-Format (einschließlich *.xpi*, *.jar* oder *.oxz*) vorliegt, muss nämlich eine Neupaketierung vorgenommen werden. Die Option *compression=xz* legt fest, dass ein *\*.orig.tar.xz* gebildet wird. Die Option *repack* ist in diesem Fall entbehrlich; aber explizit ist besser als implizit.

```
240b <OptionsWatchFile1 240b>≡ (240a)
Repacked <UpstreamPackage>.zip
RepackFlag=0
ZipSuffix=(.zip .oxz .xpi .jar)
if [[${ZipSuffix} =~ ${RecentUpstreamSuffix}]]
then
 WOpt=${WOpt}${o1f}'repack,compression=xz,'
 RepackFlag=1
fi

```

<OptionsWatchFile2 241a>

---

<sup>6</sup><https://people.debian.org/~osamu/uscan.html#WATCH-FILE-OPTIONS>

Muss eine Neupaketierung vorgenommen werden, weil aus dem Quellcode-Archiv Dateien zu entfernen sind, sollte dies im Namen der *\*.orig.tar.\**-Datei ersichtlich sein. Hierzu dient die Option *repacksuffix*. Die auszuschließenden Dateien ergeben sich aus der entsprechenden Liste (*Files-Excluded*) in der Datei *debian/copyright*.

```
241a <OptionsWatchFile2 241a>≡ (240b)
 # Excluded files
 if [-z ${RecentRepackSuffix}]
 then
 if [${RepackFlag} -eq 1]
 then
 WOpt=${WOpt}${olf}'repacksuffix=${RecentRepackSuffix}',\\n'
 else
 WOpt=${WOpt}${olf}'repack,compression=xz,\\n\
repacksuffix=${RecentRepackSuffix}',\\n'
 fi
```

<OptionsWatchFile3 241b>

Als Nächstes folgt die Option *dversionmangle*. Hiermit wird die letzte gefundene Upstream-Versionsbezeichnung in der Datei *debian/changelog* normalisiert, um sie mit der Version des verfügbaren Upstream-Archivs zu vergleichen. Dazu werden die Debian-spezifischen Suffixes wie *+dfsg* oder *+ds* im Wege der Ersetzung entfernt.

```
241b <OptionsWatchFile3 241b>≡ (241a)
 WOpt=${WOpt}${olf}'dversionmangle=s/'${RecentRepackSuffix}'//,'
 fi
```

<OptionsWatchFile4 241c>

Mit der Option *uversionmangle* werden die Zeichenketten der Dateien der Upstream-Versionen normalisiert, die aus den Links auf diese Dateien im Quellcode der Webseite extrahiert werden. In der Versionsbezeichnung werden die nicht-numerischen Zeichen (außer dem Punkt) zwecks Vereinheitlichung sinnvoll ersetzt. Damit wird die Versionsbezeichnung des Upstream-Archives versionierungsschemakonform (Kapitel 11.2, Seite 35) gestaltet. Dies wird als Versions-sortierungsindex bei der Auswahl der neuesten Upstream-Version verwendet.

```
241c <OptionsWatchFile4 241c>≡ (241b)

 # For beta-, rc- etc. releases

 WOpt=${WOpt}${olf}'uversionmangle=s/-?([^\d.]+)/~$1/;tr/A-Z/a-z/,'
 <OptionsWatchFile5 242a>
```

*filenamemangle* generiert den Upstream-Tarball-Dateinamen aus der ausgewählten *href*-Zeichenkette, wenn die Vergleichsmuster die neueste Upstream-Version aus der ausgewählten *href*-Zeichenkette extrahieren können. Andernfalls wird der Upstream-Tarball-Dateiname aus seiner vollständigen URL-Zeichenkette erzeugt und die fehlende Upstream-Version aus dem erzeugten Upstream-Tarball-Dateinamen eingesetzt.

Ohne diese Option wird der standardmäßige Upstream-Tarball-Dateiname generiert, indem die letzte Komponente der URL genommen und alles nach einem '?' oder '#' entfernt wird.

```
242a <OptionsWatchFile5 242a>≡ (241c)
 # For packages from Github
 set +e
 if echo ${DownloadUrl} | grep "github" > /dev/null
 then
 WOpt=${WOpt}${olf}'filenamemangle=s/.+\/v?(\d\S+)\.*/$1/,'
 fi
 set -e
```

<OptionsWatchFile8 242b>

```
242b <OptionsWatchFile8 242b>≡ (242a)
 # If there are no options
 if [${#WOpt} -eq 6]
 then
 WOpt='# ${WOpt}'
 WOpt=$(echo ${WOpt} | sed 's/\\/\/\\" \\/')
 fi

 echo -e ${WOpt} >> debian/watch
 }
```

<DebianWatchTemplate 239a>

*uscan* lädt die Web-Seite von der in *debian/watch* angegebenen *URL*. Dann sucht *uscan* unter Verwendung des in *debian/watch* angegebenen Suchmusters nach Hyperlinks (*hrefs*), die auf Upstream-Archive verweisen.

Ausgehend von der *URL*, mit der der Quellcode heruntergeladen wurde, wird im Programmskript definiert, wie nach einer neuen Version gesucht werden kann. Die *URL* wurde der Variablen *DownloadURL* als Wert zugewiesen (Kapitel 34.3.1, Seite 186).

```
243 <DebianWatchTemplate3 243>≡ (239b)
 set +e
 if echo ${DownloadUrl} | grep "github" > /dev/null
 then
 DownloadUrl=$(echo ${DownloadUrl} | \
 sed --expression 's/archive.*$/releases/')
 DownloadUrl=${DownloadUrl}' ./v?(\d\S+)\.tar\.gz'
 echo -e ${DownloadUrl} >> debian/watch
 fi
 set -e
 echo "A template for debian/watch was created." >> ${log}
 whiptail --title "Edit debian/watch!" \
 --msgbox "Please insert reasonable regular expressions\n \
 into debian/watch!" 15 60
 fi
 nano --linenumbers --mouse --softwrap ${GitPath}/debian/watch
 }
```

<DebianRulesTemplate 244a>

4. Mai 2025

Ein Beispiel:

```
opts=repack,compression=xz,dversionmangle=s/\+dfsg$//,\
uversionmangle=s/-Beta/~beta;/s/-rc/~rc/,\
filenamemangle=s/.*\v?(\d+\.\d+\.\d+(?:-(Beta|rc)\d+)?)\.\tar\
.gz/jax-maven-plugin-$1.tar.gz/ \
https://github.com/davidmoten/jax-maven-plugin/releases \
.*\v?(\d\S+)\.\tar.gz
```

Die Datei *debian/watch* kann, wenn man in dem Verzeichnis ist, in dem sich das Git-Repositorium befindet, mit dem Befehl *uscan -no-download -debug* geprüft werden. Die Option *-no-download* bewirkt, dass ein gefundenes, neueres Upstream-Archiv nicht heruntergeladen wird. Die Option *-debug* erzeugt einen für Menschen lesbaren Bericht, der auch den Status der internen Variablen anzeigt.

### 35.4.8. *debian/rules* - Grundlegender Aufbau

Diese Datei steuert den Ablauf des Buildprozesses.

Im Unterschied zu den anderen Dateien im Verzeichnis *debian* ist die Datei *debian/rules* als ausführbar zu kennzeichnen.

Wie jedes andere *Makefile* ist die Datei *debian/rules* aus mehreren Regeln zusammengesetzt, welche das Ziel definieren und wie diese Regeln ausgeführt werden. In der Debian-Policy, Kapitel 4.9[7] *Main building script: debian/rules* werden die Details erklärt.

#### 35.4.8.1. Erstellen der Datei

Existiert die Datei *debian/rules* noch nicht, wird sie angelegt.

Sie ist ein *Makefile* und hat daher eine entsprechende *Shebang* (*#!/usr/bin/make -f*).

```
244a <DebianRulesTemplate 244a>≡ (243)
function DebianRulesTemplate {
 # Called by DisplayDebianFiles

 # Strings for debian/rules
 str4rules="#!/usr/bin/make -f\n# -- makefile --\n"
 str4rulesdh=":\n\tdh \\\$@\n\n"

 if ! [-f ${GitPath}/debian/rules]
 then
 touch ${GitPath}/debian/rules
 echo -e ${str4rules} >> ${GitPath}/debian/rules
 <DebianRulesTemplate1 244b>
```

#### 35.4.8.2. Export von Variablen

Es wird durch den Export der Variablen *DH\_VERBOSE* und *DH\_OPTIONS*, sowie der Zuweisung entsprechender Werte die umfassendere Ausgabe der Meldungen eingeschaltet.

```
244b <DebianRulesTemplate1 244b>≡ (244a)
 echo -e "# Uncomment this to turn on verbose mode.\n" \
 >> ${GitPath}/debian/rules
 echo -e "export DH_VERBOSE=1\nexport DH_OPTIONS=-v\n" \
 >> ${GitPath}/debian/rules

 <DebianRulesTemplate2 245a>
```



Zusätzlich werden für verschiedene Pakettypen ergänzende Variablen exportiert. Diese werden in den jeweiligen Plugins definiert. Die Beschreibung für Java-Pakete findet sich in Kapitel 46.1 (Seite 383). Die Anpassung für Java-Pakete erfolgt durch die Funktion *Rules4Java*.

```
245a <DebianRulesTemplate2 245a>≡ (244b)
 if [${JavaFlag} -eq 1]
 then
 . build-gbp-java-plugin
 Rules4Java
 fi
```

<DebianRulesTemplate3 245b>

Für das Bauen der Mozilla-Erweiterungen werden auch zusätzliche Einträge benötigt. Die Beschreibung der Besonderheiten für die Mozilla-AddOns findet sich in Kapitel 48.2.2 (Seite 401).

Die Anpassung für *Webext*-Pakete erfolgt durch die Funktion *WebextRules* im entsprechenden Plugin (Kapitel 48 Seite 399).

```
245b <DebianRulesTemplate3 245b>≡ (245a)
 if [${WebextFlag} -eq 1]
 then
 WebextRules
 fi
```

<DebianRulesTemplate4 245c>

Entsprechendes gilt auch für die Python-Pakete (Kapitel 49.1 Seite 406).

```
245c <DebianRulesTemplate4 245c>≡ (245b)
 if [${PythonFlag} -eq 1]
 then
 PythonRules
 fi
```

<DebianRulesTemplate5 245d>

### 35.4.8.3. Aufruf der Debhelper

```
245d <DebianRulesTemplate5 245d>≡ (245c)
 echo -e ${str4rulesdh} >> ${GitPath}/debian/rules
```

<DebianRulesTemplate6 245e>

Für Maven-Pakete muss der Aufruf der *debhelper* ergänzt werden. Dies erfolgt durch die Funktion *Rules3MavenDH* des Maven-Plugins (Kapitel 47.5 Seite 398).

```
245e <DebianRulesTemplate6 245e>≡ (245d)
 if [${MavenPluginFlag} -eq 1]
 then
 Rules4MavenDH
 fi
```

<DebianRulesTemplate7 246a>

4. Mai 2025

Auch für die Mozilla-Erweiterungen muss der Aufruf der *debhelper* ergänzt werden. Dies erfolgt durch das *Web-Extension-Plugin* (Kapitel 48, Seite 399).

```
246a <DebianRulesTemplate7 246a>≡ (245e)
 if [${WebextFlag} -eq 1]
 then
 WebextRulesDH
 fi
```

<DebianRulesTemplate10 246b>

Auch beim Paketieren von Python-Paketen ist der Aufruf der Debhelper in der Datei *debian/rules* zu ergänzen.

```
246b <DebianRulesTemplate10 246b>≡ (246a)
 if [${PythonFlag} -eq 1]
 then
 PythonRulesDH
 fi
```

<DebianRulesTemplate11 246c>

#### 35.4.8.4. *debian/rules* - overrides

Manchmal ist es notwendig, vor oder nach der Ausführung der jeweiligen *debhelper*-Skripte, weitere Schritte auszuführen. Dazu wird das jeweilige Skript übersteuert.

Beispiel:

```
override_dh_auto_build:
 dh_auto_build -- -f org/xmlunit/pom.xml package -DskipTests
```

In der ersten Zeile des Beispiels wird das Ziel genannt, welches modifiziert werden soll. Die zweite Zeile muss mit einem *Tabulatorzeichen* in einer Breite von 8 Zeichen beginnen.

Das doppelte Minuszeichen bedeutet, dass zunächst die Parameter ausgeführt werden, die *dh\_auto\_build* normalerweise übergibt. Danach können weitere Parameter aufgeführt werden, die an das Programm übergeben werden.

*package* bezeichnet das sogenannte *goal*.

#### 35.4.8.5. Schluss der Funktion

Am Ende der Funktion erfolgt ein Eintrag in die Log-Datei. Die Datei *debian/rules* wird zum Editieren angezeigt. Schließlich wird sie als *Make*-Datei ausführbar gemacht.

```
246c <DebianRulesTemplate11 246c>≡ (246b)
 echo "debian/rules was created " >${log}

 fi
 nano --linenumbers --mouse --softwrap ${GitPath}/debian/rules

 if ! [-x ${GitPath}/debian/rules]
 then
 chmod ugo+x ${GitPath}/debian/rules
 echo "${GitPath}/debian/rules is now executable" >> ${log}
 fi
 }
```

<DebianSalsaCiTemplate 247a>

### 35.4.9. salsa-ci.yml

Auf *salsa.debian.org* wird im jeweiligen Projekt unter *Einstellungen - CI/CD - Allgemeine Pipelines - CI/CD configuration file debian/salsa-ci.yml* eingetragen, oder der entsprechende Dateiname, der sich im Verzeichnis *debian/* befindet.

Damit wird der automatische Bauprozess auch für die *Reproducible Builds* ausgelöst.

```
247a <DebianSalsaCiTemplate 247a>≡ (246c)
function DebianSalsaCiTemplate {
 # Called by DisplayDebianFiles

 # String for debian/salsa-ci.yml
 str4salsa="include:\n\
- https://salsa.debian.org/salsa-ci-team/pipeline/raw/master/salsa-ci.yml\n\
- https://salsa.debian.org/salsa-ci-team/pipeline/raw/master/pipeline-jobs.yml"

 if ! [-f ${GitPath}/debian/salsa-ci.yml]
 then
 touch ${GitPath}/debian/salsa-ci.yml
 echo -e ${str4salsa} >> ${GitPath}/debian/salsa-ci.yml
 echo "debian/salsa-ci.yml was created." >> ${log}
 fi
 nano --linenumbers --mouse --softwrap ${GitPath}/debian/salsa-ci.yml
}

<SelectChangesFile 329>
```

### 35.4.10. debian/javabuild

Die folgende Funktion ermöglicht die Erstellung einer Datei *debian/javabuild*, welche zum Bauen eines Java-Paketes ohne Build-System dienen kann.

Die Datei *debian/javabuild* enthält in jeder Zeile den Namen einer *.jar*-Datei gefolgt von einer Liste von Quellcode-Dateien oder -Verzeichnissen. Diese Datei wird von dem *javahelper*-Programm *jh\_build* eingelesen.

```
247b <DebianJavabuildTemplate 247b>≡ (296)
function DebianJavabuildTemplate {
 # Called by DisplayDebianFiles
 # For building a java package without build system (like maven)
 if [-f debian/javabuild]
 then
 if whiptail --title "Creating debian/javabuild?" \
 --defaultno --yesno "Should debian/javabuild be created?" \
 --yes-button "Yes" --no-button "No" 15 60
 then
 echo "# NameOfJarFile SourceDirToPackage" >>debian/javabuild
 echo "debian/javabuild was created" >> ${log}
 nano --linenumbers --mouse --softwrap debian/javabuild
 fi
 fi
}

<DisplayDebianChangelog 291b>
```

### 35.4.11. <Paketname>.install

Diese Datei wird benötigt, um anzugeben, wohin Dateien installiert werden sollen. Dabei ist zwingend darauf zu achten, dass als Paketname wirklich der Name des zu bauenden Binary verwendet wird.

```
248a <DisplayDebianFiles5 248a>≡ (230i)
 if [${WebextFlag} -eq 1] && [! -f ${GitPath}/debian/${PackName}.install]
 then
 WebextInstall
 fi

 nano --linenumbers --mouse \
 --softwrap ${GitPath}/debian/${PackName}.install
<DisplayDebianFiles6 248b>
```

Beispiel:

```
nc.jar usr/share/java
```

Damit werden die jeweiligen Dateien in das zukünftige Verzeichnis kopiert. Standardmäßig ist es nicht möglich, eine Datei umzubenennen, damit der Dateiname zum Beispiel der Benennung von Java-Bibliotheken in der *Debian Policy für Java* [28] entspricht.

Wie im Handbuch zu *dh\_install*<sup>7</sup> beschrieben. Dazu sind dann auch weitere Schritte notwendig.

- Das Paket muss eine Bauabhängigkeit zu *dh-exec* haben. Das Paket, das in der Datei *debian/control* angegeben werden muss, ist *dh-exec*
- Die Installationsdatei muss als ausführbar markiert sein.

### 35.4.12. <Paketname>.dirs

Diese Datei muss nicht zwingend den Namen des Binaries führen. Der Paketname kann auch komplett weggelassen werden.

```
248b <DisplayDebianFiles6 248b>≡ (248a)
 nano --linenumbers --mouse \
 --softwrap ${GitPath}/debian/${PackName}.dirs
<DisplayDebianFiles7 248c>
```

Beispiel:

```
usr/share/java
```

### 35.4.13. <Paketname>.docs

Die hier aufgeführten Dateien werden im Buildprozess vom entsprechenden *debhelper* *dh\_installdocs* in ein dazu erstelltes Verzeichnis */usr/share/docs/<Paketname>* installiert.

Die Datei *LICENSE* muss nicht aufgenommen werden. Diese wird automatisch nach */usr/share/docs/<Paketname>* installiert.

```
248c <DisplayDebianFiles7 248c>≡ (248b)
 nano --linenumbers --mouse \
 --softwrap ${GitPath}/debian/${PackName}.docs
<DisplayDebianFiles8 249a>
```

<sup>7</sup>[https://manpages.debian.org/unstable/debhelper/dh\\_install.1.de.html](https://manpages.debian.org/unstable/debhelper/dh_install.1.de.html)

**35.4.14. <Paketname>.links**

Diese Datei wird vom *dh\_link* aufgerufen.

```
249a <DisplayDebianFiles8 249a>≡ (248c)
 if [${WebextFlag} -eq 1] && [! -f ${GitPath}/debian/${PackName}.links]
 then
 WebextLinksTB
 fi

 nano --linenumbers --mouse \
 --softwrap ${GitPath}/debian/${PackName}.links
 <DisplayDebianFiles9 250>
```

**35.4.15. <Paketname>.desktop**

Dies ist erst einmal ein Beispiel:

```
249b <PaketnameDesktop 249b>≡
 [Desktop Entry]
 X-AppInstall-Package=JVerein
 X-AppInstall-Popcon=1
 X-AppInstall-Section=main

 Version=1.0
 Name=JVerein
 Comment=Administration of an Association
 Comment[de]=Vereinssoftware

 Exec=jameica
 Icon=jameica-icon
 Terminal=false
 Type=Application
 Categories=Office
 Keywords=Association;Verein
 StartupNotify=true
```

### 35.4.16. <Paketname>.manpages

Nach der Debian-Policy[7] sollte jedes Programm und jede Funktion eine entsprechende Handbuchseite haben. Diese sollte im selben Paket oder zumindest in einer Abhängigkeit enthalten sein.

Gibt es im Quellcode keine solchen Handbuchseiten, muss diese vom Maintainer erstellt und im Verzeichnis *debian/man* abgelegt werden.

Dieser Ort wird dann in die Datei *<Paketname>.manpages* eingetragen. Damit kann sie von *dh\_installman* aufgerufen werden.

Beispiel:

```
debian/man/build-gbp.1
```

### 35.4.17. <Paketname>.examples

Diese Datei wird vom *dh\_installexamples* aufgerufen.

```
250 <DisplayDebianFiles9 250>≡ (249a)
 nano --linenumbers --mouse \
 --softwrap ${GitPath}/debian/${PackName}.examples
```

```
<DisplayDebianFiles10 231a>
```

### 35.4.18. README.Debian

### 35.4.19. README.source

In dieser Datei können die Ausschlüsse (Kapitel 10.4.1.3, Seite 32) und ihre Begründungen dokumentiert werden<sup>8</sup>. Diese Datei erhält auf jeden Fall einen Eintrag, wenn mit *maven* gebaut wird. (Kapitel 47.4.5, Seite 398)

### 35.4.20. source/lintian-overrides

Manchmal erlauben die Debian-Richtlinien[7] Ausnahmen von einer Regel. Dann erzeugt *lintian* eine falsche Meldung. Gleiches gilt in den seltenen Fällen, in denen *lintian* selbst fehlerhaft ist.

In diesen Fällen kann die Datei *debian/source/lintian-overrides* genutzt werden, um eine solche Fehlermeldung zu unterdrücken. Damit wird dann das Programm fehlerfrei beendet.

Beispiel für eine solche Datei:

```
#/usr/share/lintian/overrides/<PackageName>
<PackageName>: <Lintian message>
```

Näheres zu den Lintian-Meldungen findet sich in Kapitel 28, (Seite 97).

## 35.5. Überprüfung der Dateien in *debian/* mit CmeFix

Der Befehl *cme fix dpkg* prüft die *dpkg*-Dateien, aktualisiert veraltete Parameter und wendet alle Korrekturen an.

Mit dem Parameter *-verbose* erhält man mehr Informationen darüber, was passiert. Der Parameter *-backup* erzeugt vor der Speicherung der Änderungen Backup-Dateien. Diese werden durch die Endung *.old* gekennzeichnet. Zu beachten ist, dass in diesem Fall die Langform der Optionen wirklich nur mit einem Bindestrich eingeleitet werden<sup>9</sup>.

251 <CmeFix 251>≡ (300)

```
function CmeFix {
 # Called by DisplayDebianFiles

 if whiptail --title "Check and fix with cme?" \
 --yesno "Should debian files be checked and fixed using 'cme fix'?" \
 --yes-button "Yes" --no-button "No" 15 60
 then
 set +e
 if whiptail --title "Backup?" \
 --yesno "Should the recent files be backed up (recommended)?" \
 --yes-button "Yes" --no-button "No" 15 60
 then
 cme fix --verbose --backup dpkg
 else
 cme fix --verbose dpkg
 fi
 set -e
 }
 <cmefix1 252>
```

<sup>8</sup>Debian-Policy, Kapitel 4.14 [7]

<sup>9</sup><https://manpages.debian.org/unstable/cme/cme.1p.en.html>

4. Mai 2025

Die Ausführung des Programmskriptes wird hier angehalten, damit die Ausgaben des Befehls *cme fix dpkg* analysiert werden können. In einem anderen Terminal können – sofern die Backupoption gewählt wurde – die ursprünglichen Dateien mit den neu erstellen Dateien verglichen, Backupdateien gelöscht und gegebenenfalls Korrekturen vorgenommen werden.

```
252 <cme fix 252>≡ (251)
 echo "Please check the result of cme fix!"
 echo "You can check and fix it in another terminal."
 echo "Please press RETURN to go on."
 read a
 fi
 }

<DisplayDebianFiles 229>
```



## 36. Änderungen am Upstream-Code vornehmen

Weiter geht es mit möglichen Änderungen am Quellcode (s. Kapitel 10.4.2, (Seite 32)). Sollen keine Änderungen am Quellcode vorgenommen werden, folgt direkt die Behandlung der Datei *debian/changelog* (Kapitel 37.1, Seite 291).

```
253a <BuildNewRevision5-1 253a>≡ (228)
 # Patches treatment
 PatchesTreatment

 <BuildNewRevision6 291a>
```

### 36.1. Sollen Änderungen am Upstream-Code vorgenommen werden?

Wenn eine neue Revision erstellt wird, wird der Nutzer gefragt, ob Veränderungen am Quellcode vorgenommen werden sollen. Dazu prüft das Programm zunächst, ob ein Verzeichnis *debian/patches* bereits vorhanden ist, und teilt dem Nutzer zuvor das Ergebnis dieser Prüfung mit.

Ist das Verzeichnis *debian/patches* vorhanden, erhält der Nutzer die Gelegenheit, die Anwendbarkeit der in diesem Verzeichnis vorhandenen Patches zu prüfen.

```
253b <PatchesTreatment 253b>≡ (319b)
 function PatchesTreatment {
 # Called by BuildNewRevision

 # Patches treatment
 cd ${GitPath}
 if [! -d debian/patches]
 then
 whiptail --title "Info" \
 --msgbox "There is no directory debian/patches" 15 60
 <PatchesTreatment1 254a>
```

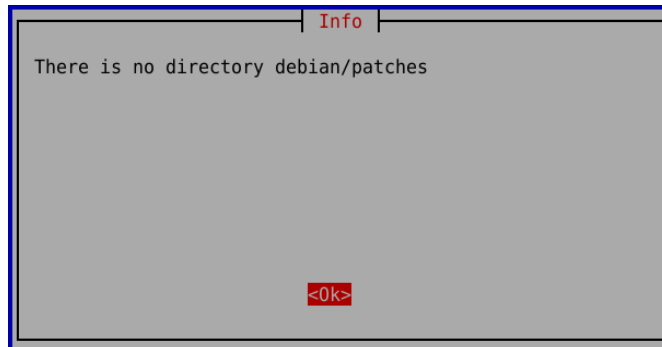


Abbildung 36.1.: Es gibt kein Verzeichnis debian/patches.

Nun kann noch entschieden werden, ob für dieses Paket Patches erstellt werden sollen (Seite 256).

254a  $\langle PatchesTreatment1\ 254a \rangle \equiv$  (253b)

```
else
 whiptail --title "Info" \
 --msgbox "There is a directory debian/patches" 15 60
```

$\langle PatchesTreatment2\ 254b \rangle$

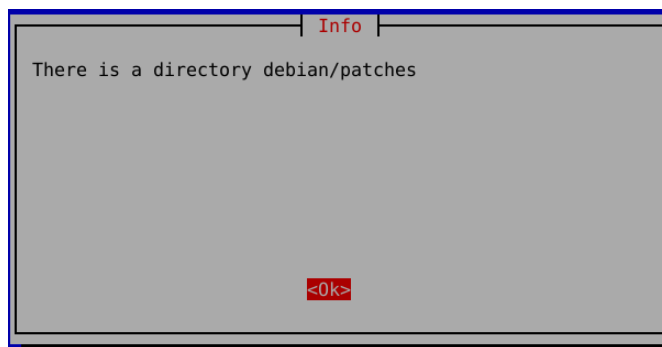


Abbildung 36.2.: Es gibt ein Verzeichnis debian/patches.

254b  $\langle PatchesTreatment2\ 254b \rangle \equiv$  (254a)

```
if whiptail --title "Applicability check"!' \
 --yesno "Do you want to check the applicability of the patches?" \
 --yes-button "Yes" --no-button "No" 15 60
then
 ApplyCheck
fi
```

$\langle PatchesTreatment2-1\ 256a \rangle$

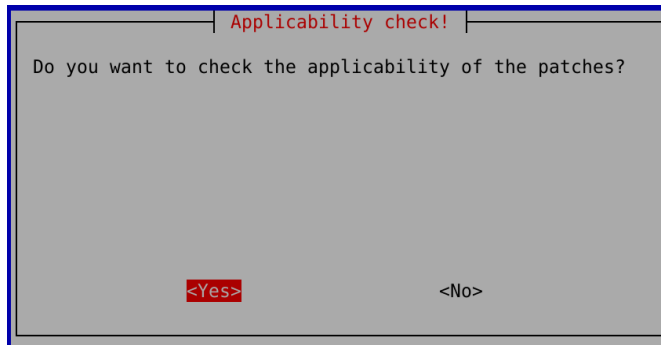


Abbildung 36.3.: Anwendbarkeit der Patches prüfen?

Die Funktion *ApplyCheck* prüft alle im Verzeichnis *debian/patches* vorhandenen Patches, unabhängig davon, ob sie in der Datei *debian/patches/series* aufgeführt sind, auf ihre Anwendbarkeit.

Die Ergebnisse werden nicht nur im Terminal ausgegeben, sondern auch zum späteren Nachlesen in die Log-Datei des jeweiligen Projektes geschrieben.

Die Ergebnisse dieses „Probelaufes“ sind vor allem dann nicht aussagekräftig, wenn mehrere Patches dieselbe Datei verändern sollen.

```

255 <ApplyCheck 255>≡ (286a)
 function ApplyCheck {
 # Called by PQMigration

 echo -e "\nResults will also be written in "${log}"\n"
 echo -e "\nResults of tests of applicability of patches\n" >> ${log}
 patchfilesa=$(ls debian/patches/)
 set +e
 for element in ${patchfilesa[*]}
 do
 if [${element} != 'series']
 then
 echo "Check ${element}"
 echo "Check ${element} >> ${log}"
 acmessage=$(patch --dry-run -p1 --verbose < debian/patches/${element})
 echo ${acmessage}
 echo -e "\n"
 echo ${acmessage} >> ${log}
 fi
 done
 set -e
 # For reading whether the patches are applicable
 echo
 echo "-- Look whether the patches are applicable --"
 echo
 echo "After reading press RETURN to go on!"
 read a
 }

```

<RebasePQBranch 268b>

Nun erfolgt die Abfrage, ob Patches erstellt, bearbeitet oder gelöscht werden sollen. Außerdem kann zwischen den beiden Methoden *quilt* (Kapitel 36.5, Seite 276) und *gbp pq* (Kapitel 36.2, Seite 257) gewählt werden.

```
256a <PatchesTreatment2-1 256a>≡ (254b)
 PMTask=$(whiptail --title "Tasks:" \
 --radiolist "Do you want to create, edit or delete patches?" 15 60 4 \
 "0" "By using quilt" off \
 "1" "By using gbp pq" off \
 "2" "No" on --cancel-button "Exit" 3>&2 2>&1 1>&3)
```

<PatchesTreatment3 256b>



Abbildung 36.4.: Patches für Debian erstellen

Natürlich kann der Bauprozess auch ohne *patches* fortgesetzt werden. Wird die Option „No“ ausgewählt, geht es mit dem Editieren der Datei *debian/changelog* weiter (Kapitel 37.1, Seite 291).

Das Patchen mit *quilt* ist die „klassische“ Methode und führt direkt zu den in das Debian-Paket (unter *debian/patches*) aufzunehmenden Dateien. Das Vorgehen mit *gbp pq* ist für alle, die gut und gerne mit *Git* arbeiten. Diese Methode sollte bei der Nutzung von *git-buildpackage* bevorzugt werden.

Mit *Exit* wird das Programmskript verlassen.

```
256b <PatchesTreatment3 256b>≡ (256a)
 if [-z "${PMTask}"]
 then
 exit
 fi
```

<PatchesTreatment4 256c>

Hat man sich zum Patchen und zu einer Methode entschieden, ruft das Programmskript nun die jeweils weiterführenden Funktionen auf.

```
256c <PatchesTreatment4 256c>≡ (256b)
 if [${PMTask} -eq 0]
 then
 PatchRunNr=0
 set +e
 PatchTasks
 set -e
```

<PatchesTreatment5 257a>

Entweder man arbeitet mit *gbp pq* oder mit *quilt*. Hat man sich für die Arbeit mit *quilt* entschieden, wird die Variable *PatchRunNr* auf 0 gesetzt und die Funktion *PatchTasks* (Kapitel 36.5, Seite 276) aufgerufen.

Arbeitet man mit *gbp pq* erfolgt der Warnhinweis, dass alles committet sein muss.

```
257a <PatchesTreatment5 257a>≡ (256c)
 elif [$PMTask -eq 1]
 then
 # Message in upper case and red
 echo -e "\n\n\033[31mEVERYTHING MUST BE COMMITTED! \033[0m \n"
 CheckGitStatus
 PQMigration
 fi
 }
```

<LastQuestionsBeforeBuild 308b>

Die Funktion *PQMigration* des Programmskriptes wird aufgerufen.

Vor dem Aufruf der Funktion *PQMigration* (und vorsichtshalber später nochmals) wird zur Prüfung die Funktion *CheckGitStatus* (Kapitel 33.4.1, Seite 170) aufgerufen.

## 36.2. Arbeiten mit *gbp pq*

Um Patches zu verwalten, kann *gbp pq* verwandt werden. Dies ist hauptsächlich für Quellpakete im Format 3.0 (*quilt*) gedacht (s. Kapitel 35.4.2, Seite 231). Die Änderung des Upstream-Quellcodes erfolgt durch Dateien im Verzeichnis *debian/patches/*.

Die Funktion *PQMigration* kann unter Verwendung dieser Patches einen Patch-Queue-Zweig erstellen, falls er noch nicht existiert. Ferner kann diese Funktion einen existierenden Patch-Queue-Zweig aktualisieren.

```
257b <PQMigration 257b>≡ (268d)
 function PQMigration {
 # Called by PatchesTreatment and itself
 # Transfers patches into patch-queue branch
 npqf=0
 cd ${GitPath}
 }
```

<PQMigration0 258>

Das Programmskript unterscheidet vier Fälle.

### 36.2.1. Fallunterscheidung

Es wird geprüft, ob es eine Datei *debian/patches/series* gibt oder nicht. Gibt es diese Datei, ermöglicht das Programmskript eine Prüfung der Situation (Kapitel 36.2.3.1, Seite 260).

Auch wird geprüft, ob es einen passenden *Patch-Queue-Branch* gibt.

Es sind vier Fälle zu unterscheiden:

1. Es gibt weder eine Datei *debian/patches/series* noch einen passenden *Patch-Queue Branch*. Dann wird mit *git checkout -b* ein neuer *Patch-Queue Branch* erzeugt und in diesen gewechselt (Seite 259).
2. Die Datei *debian/patches/series* existiert nicht, aber schon ein passender *Patch-Queue Branch*. Dann wird auf dessen Vorhandensein hingewiesen und in diesen gewechselt (Seite 259).
3. Die Datei *debian/patches/series* existiert, aber kein passender *Patch-Queue Branch*. Dann wird mit *gbp pq import* ein passender *Patch-Queue Branch* erzeugt (Seite 262). Dies scheitert, wenn nicht alle Patches der Queue anwendbar sind. Daher wird **dringend** empfohlen, die vom Programmskript ermöglichte Prüfung der Situation durchzuführen (Kapitel 36.2.3.1, Seite 260).
4. Sowohl die Datei *debian/patches/series* als auch der passende *Patch-Queue Branch* existieren (Seite 267). Dann wird *gbp pq rebase* angewandt. Daher wird **dringend** empfohlen, die vom Programmskript ermöglichte Prüfung der Situation durchzuführen (Kapitel 36.2.3.1, Seite 260).

Ein *Patch-Queue*-Zweig wird lediglich in den Fällen 1 und 3 erstellt. Andernfalls wird der schon vorhandene *Patch-Queue-Zweig* auf die neue Version aktualisiert.

### 36.2.2. Die „einfachen“ Fälle

Gibt es die Datei *debian/patches/series* nicht, ist der Wechsel in den *Patch-Queue*-Zweig unkompliziert, da sich das Problem der Anwendbarkeit der Patches nicht stellt.

```
258 <PQMigration0 258>≡ (257b)
 if [! -f debian/patches/series] # debian/patches/series does not exists
 then
 # Case 2
 # Anything is easy
 npqf=1
 set +e
 if echo $(git branch) | grep --quiet 'patch-queue/'${RecentBranch}
 then
 whiptail --title "PQ-branch exists" \
 --msgbox "Branch 'patch-queue/${RecentBranch}' exists." 15 60
 git checkout patch-queue/${RecentBranch}
 <PQMigration0-1 259a>
```

Wenn die Datei *debian/patches/series* nicht existiert, aber schon ein passender *Patch-Queue-Zweig* (**Fall 2**), erscheint als Hinweis folgende Nachricht.

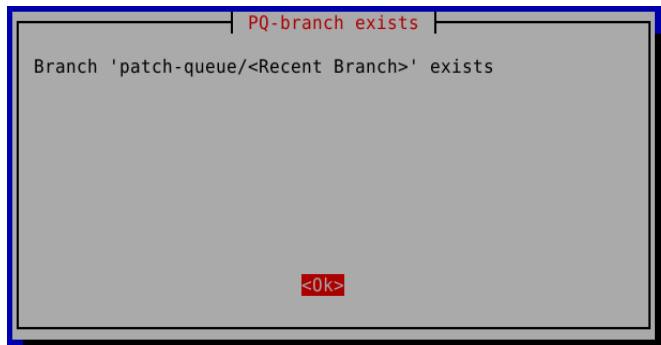


Abbildung 36.5.: Es existiert ein PQ-Zweig.

In diesen Zweig wird dann gewechselt.

Existiert die Datei *debian/patches/series* nicht und auch kein passender Patch-Queue-Zweig (**Fall 1**), wird er neu erstellt und in diesen gewechselt.

```
259a <PQMigration0-1 259a>≡ (258)
 # Case 1
 else # creating a new patch-queue branch
 git checkout -b patch-queue/${RecentBranch}
 fi
 set -e
 <PQMigration1 259b>
```

Es kann dann sofort mit der Bearbeitung des Quellcodes begonnen werden (Kapitel 36.2.4, Seite 269).

### 36.2.3. Die „schwierigeren“ Fälle

Beim Import in den Patch-Queue-Zweig und bei der Aktualisierung desselben werden die Patches auf den Upstream-Quellcode angewandt.

Voraussetzung dafür ist, dass sich alle in der Datei *debian/patches/series* aufgeführten Patches anwenden lassen. Andernfalls schlagen *gbp pq import* oder *gbp pq rebase* fehl. Dann wird kein *Patch-Queue-Zweig* erzeugt.

Sowohl *gbp pq rebase* als auch *gbp pq import* (**Fälle 3 und 4**) setzen also voraus, dass es im aktuellen Git-Zweig keine unversionierten Dateien gibt **und** sich alle bisherigen Patches anwenden lassen. Andernfalls sind Merge-Konflikte zu lösen. Hierauf wird der Nutzer hingewiesen.

```
259b <PQMigration1 259b>≡ (259a)
 else # debian/patches/series exists
 Notice="All patches listed in 'debian/patches/series' \n\
 have to be applicable'''\n\
 Otherwise you have to solve 'merge conflicts'''\
 if whiptail --title "Attention please'''\ \
 --yesno "${Notice} Do you want to check the situation?" \
 --yes-button "Yes" --no-button "No" 15 60
 <PQMigration1-0 260>
```

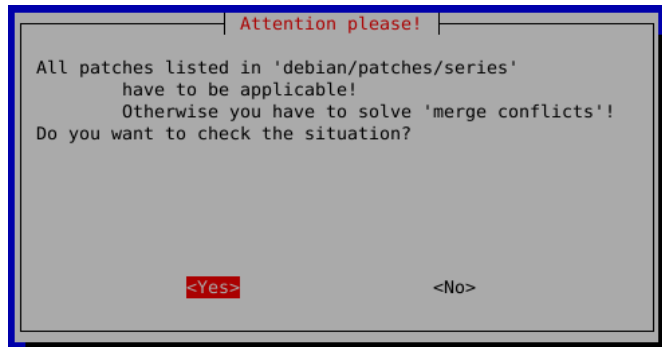


Abbildung 36.6.: Alle Patches müssen anwendbar sein.

### 36.2.3.1. Prüfen der Situation

Wird die Frage, ob die Situation geprüft werden soll, bejaht, wird – wenn gewünscht – die Funktion *ApplyCheck* aufgerufen (Seite 255). Diese prüft mit „Probelaufen“, ob die im Verzeichnis *debian/patches* vorhandenen Patch-Dateien auf den neuen Upstream-Quellcode anwendbar sind und teilt das Ergebnis mit.

Die Ergebnisse sind vor allem dann nicht aussagekräftig, wenn mehrere Patches dieselbe Datei verändern sollen.

Der Nutzer wird gefragt, ob diese Prüfung (noch einmal) erfolgen soll.

260 `<PQMigration1-0 260>≡` (259b)

```

then
 if whiptail --title "Applicability check"!' ' \
 --yesno "Do you want to check the applicability of the patches\n\
 (another time)?" \
 --yes-button "Yes" --no-button "No" 15 60
 then
 ApplyCheck
 fi
 ShowPatches "check"
<PQMigration1-1 261>

```



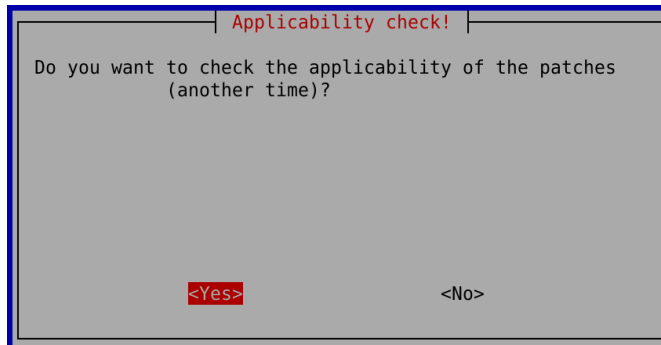


Abbildung 36.7.: (Nochmalige) Prüfung der Anwendbarkeit der Patches?

Das Programm verbleibt zunächst im ursprünglichen Zweig.

Es können Änderungen an Patch-Dateien vorgenommen werden. Dies sollte in einfachen und eindeutigen Fällen – wie folgt – geschehen. Die Funktion *ShowPatches* (Kapitel 36.4, Seite 274) zeigt die vorhandenen Patch-Dateien zur Auswahl an. Ausgewählte Patch-Dateien können dann editiert werden. In der Regel kann dies durch *Cancel* übergegangen werden,

In der Datei *debian/patches/series* werden alle Patches in der Reihenfolge aufgelistet, in der diese bisher angewandt wurden.

Wenn vorhandene Patches nicht anwendbar sind, müssen diese vor dem Wechsel in den *Patch-Queue*-Zweig in der Datei *debian/patches/series* deaktiviert (auskommentiert) werden.

Dafür wird gefragt, ob die Datei *debian/patches/series* editiert werden soll.

```
261 <PQMigration1-1 261>≡ (260)
 if whiptail --title "Attention please"'\ \
 --yesno "Do you want to edit 'debian/patches/series'?" \
 --yes-button "Yes" --no-button "No" 15 60
 then
 nano --linenumbers --mouse --softwrap debian/patches/series
 fi
fi

echo -e ${Notice}
CheckGitStatus

fi
<PQMigration2 268a>
```

Abbildung 36.8.: Soll *debian/patches/series* editiert werden?.

Nicht-anwendbare Patches sollten bereits hier darauf geprüft werden, ob die Probleme durch Upstream bereits behoben wurden und diese Patches damit entfallen.

Dann können diese Patches aus der Datei *debian/patches/series* entfernt werden.

Werden Patches oder die Datei *debian/patches/series* verändert, müssen diese Änderungen committet werden. Daher wird auch nochmals der *Git*-Status geprüft.

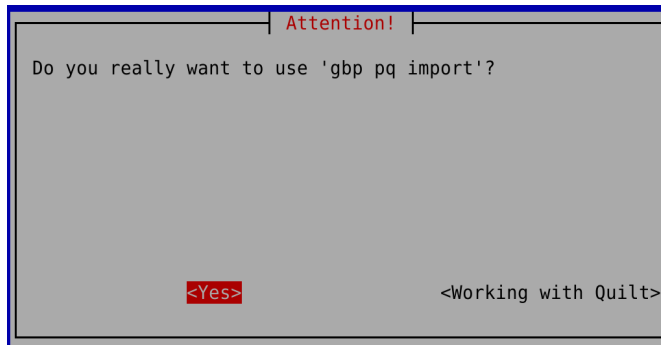
### 36.2.3.2. *gbp pq import*

Existiert die Datei *debian/patches/series*, aber noch kein entsprechender Patch-Queue-Zweig, wird *gbp pq import* ausgeführt. Dies geschieht in der Funktion *PQImport* (**Fall 3**).

```

262 <PQMigration3 262>≡ (268a)
 else # patch-queue branch does not exist
 # Case 3
 if whiptail --title "Attention"''!'' \
 --yesno "Do you really want to use 'gbp pq import'?" \
 --yes-button "Yes" --no-button "Working with Quilt" 15 60
 then
 PQImport 0
 <PQMigration3-1 276a>

```

Abbildung 36.9.: Soll *gbp pq* genutzt werden?

*gbp pq import* erzeugt einen neuen Patch-Queue-Zweig aus den *quilt patches* im Verzeichnis *debian/patches*, die in der Datei *debian/patches/series* aufgeführt sind. Diese Patches müssen ohne Unschärfe anwendbar sein.

263 `<PQImport 263>` ≡ (171)

```
function PQImport {
 # Called by PQMigration and itself
 # returnflag=$1
 # if [! ${returnflag}]
 # then
 # returnflag=1
 # fi
 cd ${GitPath}
 set +e
 if echo $(git branch) | grep --quiet 'patch-queue/'${RecentBranch}
 # patch-queue branch already exists
 then
 set -e
 return
 fi
 set -e

 if [-f debian/patches/series]
 # debian/patches/series exists
 then
 if whiptail --title "There are patches" \
 --yesno "Do you like to import the current patches\n\
 onto the patch-queue branch? (recommended)" \
 --yes-button "Yes" --no-button "No" 15 60
```

`<PQImport1 264>`

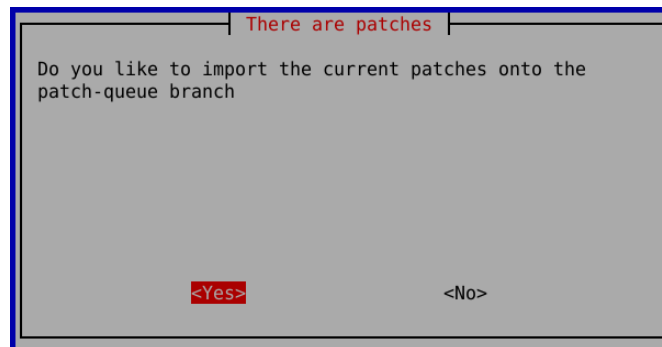


Abbildung 36.10.: Es gibt Patches

```
264 <PQImport1 264>≡ (263)
 then
 CheckGitStatus
 set +e
 echo "Notice from gbp pq import: " >> ${log}
 gbp pq --verbose import >> ${log} 2>&1
 if [$? -eq 1]
 then
 Notice="\n\nAll patches listed in debian/patches/series\n\
 have to be applicable''!''\n\
 For Details, look into the log file of the project.\n"
 FailureNotice ${Notice}
 <PQImport2 265>
```

```

All patches listed in debian/patches/series
have to be applicable!
For Details, look into the log file of the project.

Break for fixing it in another terminal
After fixing press RETURN to go on!

```

Abbildung 36.11.: PQ-Import fehlgeschlagen

Schlägt der Import fehl, wird eine Möglichkeit zur Fehlerbehebung eröffnet (Kapitel 33.4.2, Seite 171).

Zur Fehlerbehebung nach einem Fehlschlag können nur allgemeine Hinweise gegeben werden.

In einem weiteren Terminal können folgende Schritte ausgeführt werden.

- In der Log-Datei den fehlgeschlagenen Patch identifizieren.
- Einfache Fehler direkt im Patch-File anpassen;
- Patch-File committen.
- Alternativ Patch in der Datei *debian/patches/series* deaktivieren;
- dann *debian/patches/series* committen.
- Zurück in das ursprünglichen Terminal;
- dort mit <RETURN> weiter.

Nach der Fehlerbehebung kann ein erneuter Importversuch unternommen werden. Dieser Vorgang kann wiederholt werden, um sich Schritt für Schritt dem erfolgreichen Import zu nähern.

Wird der Versuch der Fehlerbehebung endgültig für misslungen erachtet, kann der Import abgebrochen werden.

```

265 <PQImport2 265>≡ (264)
 if whiptail --title "Fixed?" --yesno "Retry?" \
 --yes-button "Yes" --no-button "No import" 15 60
 <PQImport3 266a>

```

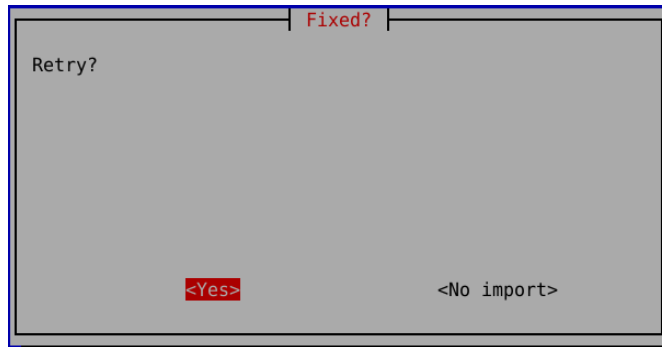


Abbildung 36.12.: Fixed? Retry?

266a `<PQImport3 266a>≡` (265)

```

then
 PQImport ${returnflag}
else
 whiptail --title "No import onto a patch-queue branch" \
 --msgbox "Let's go on without the import\n\
of the current patches onto a patch-queue branch" \
 15 60
 fi
fi
set -e
fi
}

```

`<ClassicalOrUscan 183>`

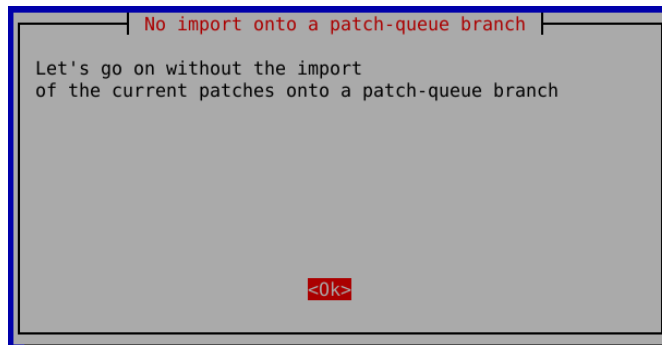


Abbildung 36.13.: Kein Import in Patch-Queue

Ziel ist es, durch `gbp pq import` einen *Patch-Queue*-Zweig zu erstellen, in dem dann gearbeitet werden kann.

266b `<PQImport4 266b>≡`

```

else
 whiptail --title "Done" \
 --msgbox "Imported the current patches onto the patch-queue branch" \
 15 60
 <PQImport5 (nicht definiert)>

```

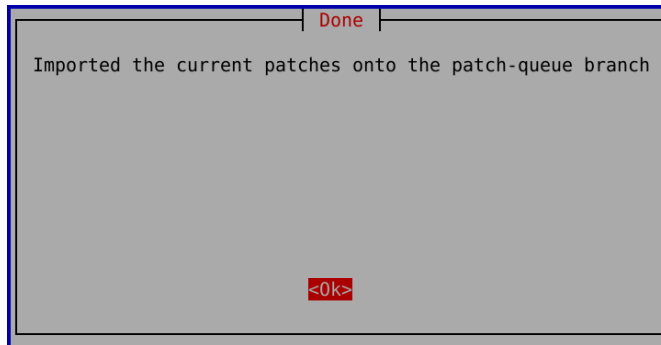


Abbildung 36.14.: PQ-Import erfolgreich

Der erfolgreiche Import in den PQ-Branch wird mit einem Dialog angezeigt.

Der Inhalt von *debian/patches* wird mit *gbp pq* in den Patch-Queue-Branch importiert und in diesen gewechselt<sup>1</sup>.

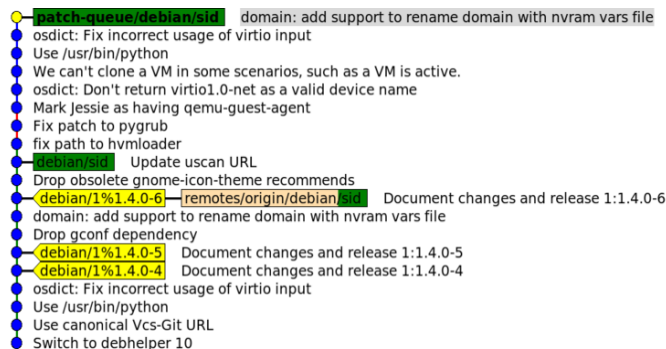
Nach erfolgreichem Import in den *Patch-Queue*-Zweig kann mit der Bearbeitung des Quellcodes begonnen werden (Kapitel 36.2.4, Seite 269).

Beim Import wird eine Ausgabe wie folgt erzeugt:

```
gbp:info: Versuchen, Patches \
unter 'aaa1011bfd5aa74fea43620aae94709de05f80be' \
anzuwenden.
```

```
gbp:info: 18 Patches, die in 'debian/patches/series' \
aufgelistet sind, importiert unter 'patch-queue/debian/sid'.
```

Es wurde mit *gbp pq* jede Patchdatei importiert und in den neu erstellten Patch-Queue-Zweig *patch-queue/debian/sid* gewechselt.

Abbildung 36.15.: Der Patch-Queue-Zweig mit Patches von *debian/patches* wurde angewandt.[3]

### 36.2.3.3. Aktualisieren des Patch-Queue-Zweiges

Das Programmskript prüft, ob bereits ein entsprechender Patch-Queue-Zweig existiert.

Existiert ein passender Patch-Queue-Zweig ruft das Programmskript zunächst die Funktion *RebasePQBranch* auf (**Fall 4**).

<sup>1</sup>s. Abschnitt *gbp.patches.html* und *man.gbp.pq.html*[3]

Andernfalls wird Patch-Queue-Zweig durch *gbp pq import* aus dem aktuellen *Git*-Zweig erstellt (Kapitel 36.2.3.2, Seite 266).

268a `<PQMigration2 268a>≡` (261)

```

if [${npqf} -eq 0]
then
 set +e
 if echo $(git branch) | grep --quiet 'patch-queue/'${RecentBranch}
 # patch-queue branch exists
 then
 # Case 4
 RebaseCounter=0
 RebasePQBranch

```

`<PQMigration3 262>`

Die Funktion *RebasePQBranch* führt ein *gbp pq rebase* aus. Die Commits des *debian/-* Zweiges werden auf den Patch-Queue-Zweig angewandt.

Mit *gbp pq rebase* wird in den Patch-Queue-Zweig gewechselt, der mit dem aktuellen Zweig verbunden ist. Alle Neuerungen des aktuellen Zweiges werden durch ein *rebase* in den Patch-Queue-Zweig übernommen.[46]

268b `<RebasePQBranch 268b>≡` (255)

```

function RebasePQBranch {
 # Called by PQMigration and itself
 if [${RebaseCounter} == 0]
 then
 gbp pq rebase --verbose
 else
 git rebase --continue
 fi
}

```

`<RebasePQBranch1 268c>`

Misslingt die Ausführung von *gbp pq rebase*, kann und muss händisch eingegriffen werden, um die Situation zu bereinigen

Hierzu werden von *Git* dem Nutzer Hinweise gegeben, die sorgfältig studiert und meist befolgt werden sollten.

268c `<RebasePQBranch1 268c>≡` (268b)

```

if [$? -ne 0]
then
 git rebase --show-current-patch | cat

```

`<RebasePQBranch2 268d>`

An dieser Stelle wird der Patch angezeigt, der nicht (komplett) angewandt werden kann. Mit dieser Hilfe ist es dann möglich, nach der Unterbrechung manuell die Merge-Konflikte aufzulösen.

268d `<RebasePQBranch2 268d>≡` (268c)

```

Notice="gbp pq rebase failed"'\n\
All changes must be committed"'\n\
All patches have to be applicable"'\n\
FailureNotice ${Notice}
RebaseCounter=$(expr ${RebaseCounter} + 1)
RebasePQBranch

```

```

fi
}

```

`<PQMigration 257b>`



Wie die Konflikte aufgelöst werden können, wird im Folgenden beschrieben.

Das Programmskript gibt in solchen Fällen den Hinweis, auf ein weiteres Terminal zu wechseln. Dort kann mit `git status` überprüft werden, dass ein unvollständiger Rebase vorliegt.

Mit dem nachstehenden Befehl wird in dem weiteren Terminal der fehlgeschlagene Patch angezeigt.

```
git rebase --show-current-patch
```

In noch einem weiteren Terminal kann das (noch) Vorhandensein der zu patchenden Datei mit folgendem Befehl geprüft werden.

```
tar --list --file ../<UpstreamPackageName>.orig.tar.xz | \
grep <Filename>
```

Ist die Datei nicht (mehr) vorhanden, kann der Patch mit

```
git rebase --skip
```

entfernt werden. Sodann ist im (ursprünglichen) Terminal, in dem das Programmskript läuft, mit `RETURN` fortzufahren. Damit wird zunächst ein `git rebase -continue -verbose` ausgeführt.

Wenn die zu patchende Datei existiert, muss sie in einem Editor bearbeitet werden. Die Patches sind so zu verändern, dass sie auf die neue Version angewandt werden können.

Die Datei ist dann zu speichern und ein `git add <Dateiname>` auszuführen. Im ursprünglichen Terminal ist mit `RETURN` fortzufahren.

Der Vorgang ist gegebenenfalls sooft zu wiederholen, bis das Programmskript mit seiner Ausführung fortfährt.

Eine Möglichkeit ist es auch, durch den Befehl

```
gbp pq import --time-maschine=<n>
```

solange Commit für Commit durchzugehen, bis die Patch-Queue angewandt werden kann.

`n` gibt dabei die maximale Anzahl der Rückschritte an.

Danach kann ein erneuter Versuch erfolgen.

Notfalls kann der Prozess mit

```
git rebase --abort
```

abgebrochen werden.

Im Erfolgsfalle geht es mit der Bearbeiten des Quellcodes weiter.

### 36.2.4. Bearbeiten des Quellcodes

Dass mit der Bearbeitung des Quellcodes im *Patch-Queue*-Zweig begonnen werden kann, zeigt das Programmskript im Terminal an.

269 `<PQMigration4 269>≡`

(276a)

```
Starting the work in the patch-queue branch
echo
echo "\033[32m-- Starting the work in the patch-queue branch --/033[0m"
echo
echo "Break for patching in another terminal"
```

4. Mai 2025

```
echo "Do not forget to commit the changes!"
echo
echo "After finishing press RETURN to go on!"
read a
```

*(PQMigration5 272a)*

```
-- Starting the work in the patch-queue branch --
Break for patching in another terminal
Do not forget to commit the changes!
After finishing press RETURN to go on!
```

Abbildung 36.16.: Unterbrechung zum Patchen

Nach der Bearbeitung des Quellcodes kann das Programmskript mit einem *RETURN* fortgesetzt werden. Dann erfolgt der Export der Patches (Kapitel 36.2.5, Seite 272).

Im separaten Terminal sollte mit *git branch -v* festgestellt werden, dass der richtige Git-Zweig (*patch-queue/-Zweig*) aktiv ist und dass dessen Stand dem des bisherigen Git-Zweiges entspricht. Sodann können Dateien im Patch-Queue-Zweig bearbeitet werden.

Code kann hinzugefügt, verändert oder entfernt werden. So können die Patches erstellt oder „in Form gebracht“ werden.

Dabei sind die Änderungen möglichst kleinteilig zu committen. Was später ein einzelner Patch in *debian/patches/* werden wird, wird einfach durch einen Commit hinzugefügt.

Die erste Zeile der Commit-Nachricht wird später Teil des Patch-Namens. Die folgenden Zeilen enthalten die Details zu den Funktionen des Patches. Daher ist es sinnvoll mehrzeilige Commit-Nachrichten zu schreiben.

Die mehrzeilige Commit-Nachricht sollte dann im Patch der *Patch tagging guideline* entsprechen<sup>2</sup> Der Header der Patch-Datei sieht dann wie folgt aus:

Der Autor und das Datum werden automatisch gesetzt. Der unter *Gbp-PQ* anzugebene Name darf keine Leerzeichen enthalten. Diese sind gegebenenfalls durch *Unterstriche* (  ) zu ersetzen. Dies wird der Name der Patch-Datei im Verzeichnis *debian/patches*.

```
From: Autor <E-Mail-Adresse>
Date: <Erstellungsdatum und -zeit>
Subject: <Commit-Nachricht>
```

```
Gbp-Pq: Name <name of the patch>
Forwarded: Yes/No <if necessary>
Last Update: <date of the last version of the patch>
```

```
* posix/regcomp.c (re_compile_fastmap_iter): Rewrite COMPLEX_BRACKET
 handling.
```

```
Origin: upstream, http://sourceware.org/git/?p=glibc.git;\
a=commitdiff;h=bdb56bac
Bug: http://sourceware.org/bugzilla/show_bug.cgi?id=9697
Bug-Debian: http://bugs.debian.org/510219
```

Diese kann mit *git commit --amend* korrigiert werden. Es öffnet sich der Editor zur Eingabe der neuen Commit-Message.

Die Bearbeitung des Quellcodes sollte damit beginnen, die in der Datei *debian/patches/series* deaktivierten Patches anwendbar zu machen. Hierzu sind die Patch-Datei und die zu patchende Datei zu vergleichen.

Stellt sich dabei heraus, dass die Probleme durch die Upstream-Entwickler bereits behoben worden sind, ist (an dieser Stelle) nichts mehr zu tun.

<sup>2</sup>[https://dep-team.pages.debian.net/deps/dep3/\[26\]](https://dep-team.pages.debian.net/deps/dep3/[26])

### 36.2.5. Export der Patches

Sobald wir mit den Commits zufrieden sind, können die Patches in *debian/patches/* unter Verwendung von *gbp pq* neu generiert werden. Dadurch wechseln wir zurück zum Zweig *debian/sid* und regenerieren die Patches mit einer Methode ähnlich dem *git-format-patch*:

Das Ergebnis könnte nun wie folgt hinzugefügt werden:

```
gbp pq export
git add debian/patches
git commit
```

*gbp pq export* bedeutet:

Exportiert die Patches auf dem Patch-Queue-Branch, der mit dem aktuellen Zweig verbunden ist, in eine Quilt-Reihe von Patches nach *debian/patches/* und aktualisierte die Datei *series* im aktuellen Zweig (z.B. *debian/sid*)<sup>3</sup>[3].

Damit das Ergebnis nicht jedes Mal von Hand übertragen werden muss, kann auch *-commit* an den *gbp-Export*befehl übergeben werden.

```
272a <PQMigration5 272a>≡ (269)
 # Export
 if whiptail --title "Use gbp pq export?" \
 --yesno "Do you like to use 'gbp pq export --commit'?" \
 --yes-button "Yes" --no-button "No" 15 60
 <PQMigration7 272b>
```



Abbildung 36.17.: Soll *gbp pq export* angewandt werden?

```
272b <PQMigration7 272b>≡ (272a)
 then
 CheckGitStatus
 gbp pq export --commit --verbose >> ${log} 2>&1
 echo "Check branch."
 git branch --verbose
 echo "Press RETURN to continue!"
 read a
 <PQMigration8 273>
```

<sup>3</sup><https://honk.sigxcpu.org/projects/git-buildpackage/manual-html/gbp.patches.html>

Nach dem *RETURN* kann die Datei *debian/changelog* aktualisiert werden (z. B. durch Ausführen von *gbp dch -S -a*, was dasselbe ist wie *gbp dch --snapshot --auto*. (Kapitel 37.1, Seite 291)

Das Paket kann dann – wie gewohnt – erstellt werden.

Wird die Frage verneint, kann manuell eingegriffen werden.

```
273 <PQMigration8 273>≡ (272b)
 else
 git log
 git checkout ${HistoricBranch}
 ReplaceConfigLines 'RecentBranch' ${HistoricBranch}
 git branch
 echo "-- You have cancelled the export from patch-queue branch --"
 echo
 echo "You are still in the patch-queue branch!"
 echo "Break for fixing in another terminal"
 echo "After finishing press RETURN to go on!"
 read a
 fi
}

<PatchTasks 277b>
```

### 36.3. Bauen schlägt fehl

Wenn das Bauen fehlschlägt und weitere Patches benötigt werden oder Patches zu korrigieren sind, muss das Programmskript neugestartet werden.

Vor einem Neustart muss gegebenenfalls das Bau-Verzeichnis bereinigt werden. Dazu wird es auf den Stand nach dem Export gesetzt.

Im *Patch-Queue*-Zweig können dann Änderungen an den Quellcode-Dateien vorgenommen werden. Der danach notwendige Export der Patches wird in Kapitel 36.2.5 (Seite 272) beschrieben.

Es kann die Patch-Queue geprüft und eventuell auch mittels *dquilt* bearbeitet werden. Dann kann im Hauptzweig ein erneuter Bau-Versuch unternommen werden.

### 36.4. Auswahl der Patches

Die folgende Funktion *ShowPatches* kann sowohl beim Arbeiten mit *gbp pq* als auch beim Arbeiten mit *quilt* Anwendung finden.

Beim Aufruf durch die Funktion *PQMigration* wird die Funktion *ShowPatches* mit dem Parameter *check* aufgerufen.

Die einzelnen Patches können angepasst werden. Wenn benötigt, kann man sich die Ausgaben des „Probelaufes“ in der Log-Datei ansehen.

```
274 <ShowPatches 274>≡ (278)
function ShowPatches {
 # Called by EditPatch DeletePatch PQMigration and itself
 actionstr=${1}

 patchfilesa=$(ls debian/patches)
 i=0; slct=''
 for element in ${patchfilesa[*]}
 do
 if [${element} != 'series']
 then
 slct=${slct} '$i' '${element}' off '
 newPFA[$i]=${element}
 i=$(expr $i + 1)
 fi
 done

 set +e
 PatchFileNo=$(whiptail --title "Select patch" \
 --radiolist "Select one of these patches for editing.\
 In doubt press Cancel." \
 --cancel-button "Cancel" 15 80 8 \
 $slct 3>&2 2>&1 1>&3)

 <ShowPatches1 275a>
```

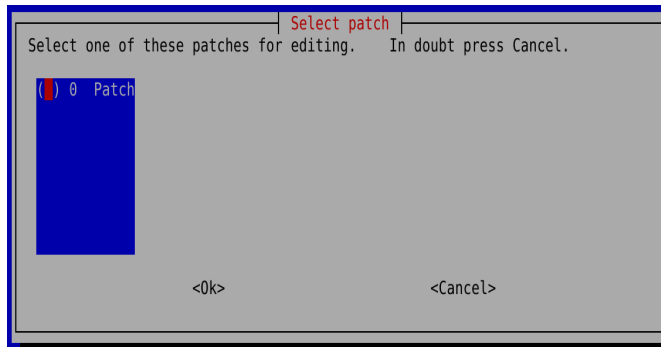


Abbildung 36.18.: Dateiauswahl bestätigen

```
275a <ShowPatches1 275a>≡ (274)
 if [$? -eq 1]
 then
 return
 fi
 set -e
```

```
 if ["${actionstr}" = "check"]
 then
 # Code for PQMigration
 if [-z "${PatchFileNo}"]
 then
 return
 fi
 PatchFileName=${newPFA[${PatchFileNo}]}
 nano --linenumbers --mouse --softwrap debian/patches/${PatchFileName}
 ShowPatches "check"
```

<ShowPatches2 275b>

Der ausgewählte Patch wird zum Editieren angezeigt. Alle Änderung sind zu committen.

```
275b <ShowPatches2 275b>≡ (275a)
```

```
 else
 if [-z "${PatchFileNo}"]
 then
 PatchFileName=""
 else
 PatchFileName=${newPFA[${PatchFileNo}]}
 if ! whiptail --title "${PatchFileNo}" \
 --yesno "Do you want to ${actionstr} ${PatchFileName}?" \
 --yes-button "Yes" --no-button "No" 15 60
 then
 ShowPatches
 fi
 fi
 fi
 }
}
```

<EditPatch 290>

## 36.5. Nutzung von Quilt

Die Änderungen am Upstream können auch mittels Quilt erfolgen.

```
276a <PQMmigration3-1 276a>≡ (262)
 else
 PatchTasks # Working with Quilt
 fi
 fi
 set -e
 fi
 <PQMmigration4 269>
```

Wie zuvor beschrieben, ist *dquilt* zunächst einzurichten (Kapitel 19.6, Seite 69).

Die Einrichtung in der *.bashrc* lässt sich bei der Ausführung dieses Programms nicht nutzen (Kapitel 19.6, Seite 69).

Das Programm prüft daher zunächst, ob *quilt* zur Verfügung steht. Da das Skript den Alias der *.bashrc* unbeachtet lässt, wird *dquilt* in einer Variablen definiert.

```
276b <CreateDquilt 276b>≡ (285b)
 function CreateDquilt {
 # Called by PatchTasks

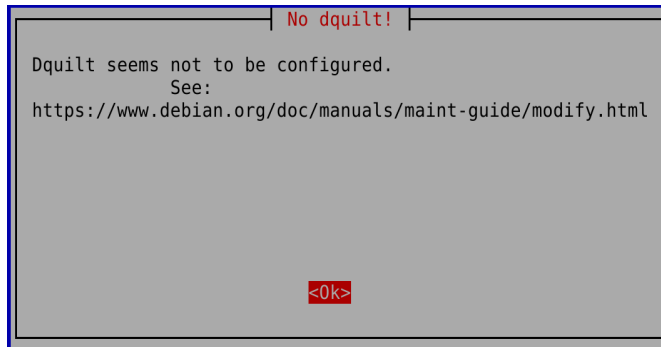
 # Check whether quilt is available
 if [${PatchRunNr} -eq 0]
 then
 if [! -f ~/.quiltrc-dpkg]
 <CreateDquilt1 276c>
```

Die Zeile prüft, ob es im Home-Verzeichnis des Nutzers eine Datei *.quiltrc-dpkg* existiert. Wenn diese Datei nicht existiert, erfolgt die Meldung, dass *dquilt* nicht konfiguriert ist, weil dann der entsprechende Alias für *quilt -quiltrc=\$HOME/.quiltrc-dpkg* nicht erstellbar ist.

```
276c <CreateDquilt1 276c>≡ (276b)
 then
 whiptail --title "No dquilt!" \
 --msgbox "Dquilt seems not to be configured.\n \
 See: https://www.debian.org/doc/manuals/maint-guide/modify.html" \
 15 62
 exit
 fi

 <CreateDquilt2 277a>
```



Abbildung 36.19.: *dquilt* ist nicht konfiguriert.

Wenn die Datei *.quiltrc-dpkg* nicht gefunden wird, gibt das Skript eine entsprechende Meldung aus und wird beendet.

Nur wenn die Datei *./quiltrc-dpkg* existiert, kann die folgende Definition erfolgen.

```
277a <CreateDquilt2 277a>≡ (276c)
 # Definition of dquilt
 dquilt="quilt --quiltrc=${HOME}/.quiltrc-dpkg"
 fi
}

<MakePatches 278>
```

### 36.5.1. Bisher kein Patch vorhanden

Zunächst prüft das Programmskript, ob das Verzeichnis *debian/patches* vorhanden ist.

```
277b <PatchTasks 277b>≡ (273)
 function PatchTasks {
 # Called by PatchesTreatment and itself

 cd ${GitPath}
 if [! -d debian/patches]
 then
 # Create patch
 if whiptail --title "Patches" --yesno "Is a patch necessary?" \
 --yes-button "Yes" --no-button "No" 15 60
 then
 CreateDquilt
 MakePatches
 else
 return
 fi
 fi

 <PatchTasks0 279a>
```

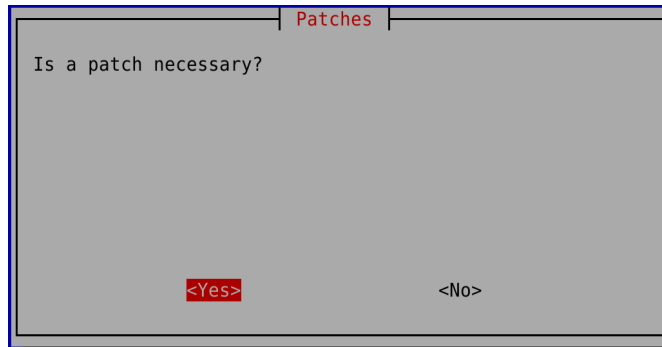


Abbildung 36.20.: Wird ein Patch benötigt?

Gibt es bisher keinen Patch, kann nur ein neuer Patch erstellt werden. Dies erfolgt mit der Funktion *CreateNewPatch*. (Kapitel 36.5.3, Seite 280).

```
278 <MakePatches 278>≡ (277a)
 function MakePatches {
 # Called by PatchTasks and itself

 cd ${GitPath}
 cnpr=0
 CreateNewPatch
 if [$cnpr -ne 0]
 then
 return
 fi

 PatchRunNr=1

 if whiptail --title "Another patch?" \
 --yesno "Do you want to apply another patch?" --yes-button "Yes" \
 --no-button "No" 15 60
 then
 MakePatches
 fi
 }

<ShowPatches 274>
```

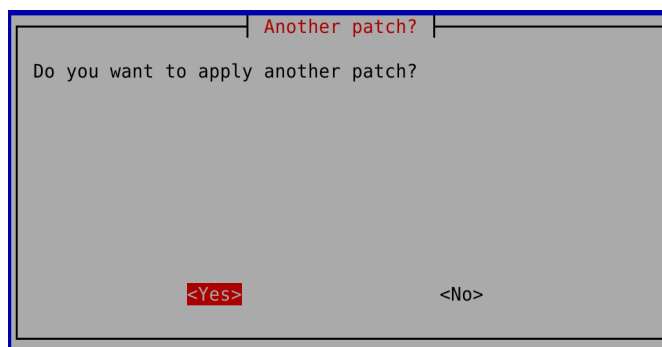


Abbildung 36.21.: Ein weiterer Patch?

### 36.5.2. Was soll getan werden?

Sind bereits Patches vorhanden, wird zur Auswahl gestellt, ob weitere Patches erzeugt, editiert oder entfernt werden sollen. Diese Aufgaben können in beliebiger Reihenfolge und auch mehrmals ausgeführt werden.

```
279a <PatchTasks0 279a>≡ (277b)
 else
 CreateDquilt
 PTask=$(whiptail --title "Tasks:" \
 --radiolist "What do you like to do? " 15 60 8 \
 "0" "Display patch files to check or edit them" off \
 "1" "Create additional patch" off \
 "2" "Add another patch to existing patch file" off \
 "3" "Show patch files for deleting" off \
 "4" "Edit debian/patches/series" off \
 "5" "Exit to go on" on --cancel-button "Cancel" 3>&2 2>&1 1>&3)
<PatchTasks1 279b>
```

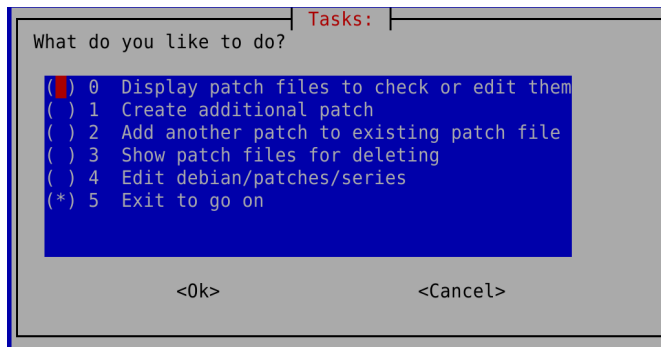


Abbildung 36.22.: Aufgaben für Quilt

Wird der Button *Cancel* gedrückt, so wird ebenfalls die Aufgabe *Exit to go on* ausgeführt. Es wird der Ausgangszustand wieder hergesellt. (Kapitel 36.5.7, Seite 287)

```
279b <PatchTasks1 279b>≡ (279a)

 if [$? -eq 1]
 then
 PTask=5
 fi
```

<PatchTasks2 289a>

Im folgenden Abschnitt wird den einzelnen Aufgaben Befehle zugeordnet. Dabei enthält die Variable *PTask* eine Ziffer von 0 bis 4.

Wenn beispielsweise dieser Variablen der Wert *0 = Display patch files to check or edit them* zugewiesen wurde, dann wird die Funktion *ChangePatches* ausgeführt.

In der folgenden *case-Anweisung* ist *0*) ein Wert aus der Liste von 0 bis 4, gegen den der Inhalt der Variable *PTask* verglichen wird.

```
279c <PatchTasks3 279c>≡ (289a)
 1) MakePatches;; # If (more) patches are necessary
<PatchTasks4 281b>
```

### 36.5.3. Neuen Patch erstellen

Beim Erstellen eines neuen Patches wird zunächst der Name der Patch-Datei erfragt.

```
280a <CreateNewPatch 280a>≡ (282)
function CreateNewPatch {
 # Called by MakePatches

 PatchFileName=$(whiptail --title "Patch name" \
 --inputbox "Name of the patchfile:" \
 --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
 if [$? -ne 0]
 then
 cnpr=1
 return 1
 fi

 <CreateNewPatch0 280b>
```

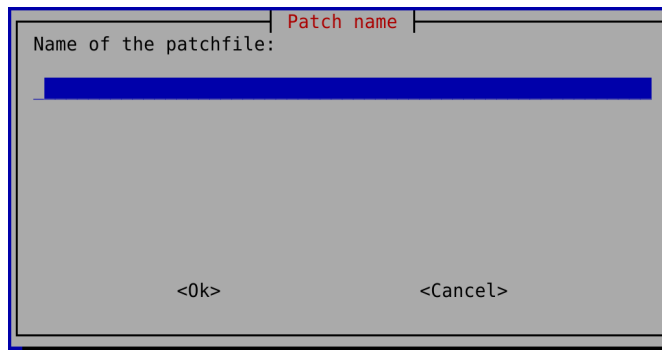


Abbildung 36.23.: Name des Patches

```
280b <CreateNewPatch0 280b>≡ (280a)

 # Because a user might use blanks in a filename
 PatchFileName=$(echo ${PatchFileName} | sed --expression='s/ /-/g')
 if [-z "${PatchFileName}"]
 then
 cnpr=1
 return 1
 fi
```

<CreateNewPatch1 280c>

Es wird geprüft, ob ein gleichnamiger Patch bereits existiert.

```
280c <CreateNewPatch1 280c>≡ (280b)
 if [-f debian/patches/${PatchFileName}]
 then
 if ! whiptail --title "Patch exists" \
 --yesno "${PatchFileName} exists already.\nContinue?" \
 --yes-button "Yes" --no-button "No" 15 60
 then
 return 1
 fi
 fi
```

<CreateNewPatch2 281a>

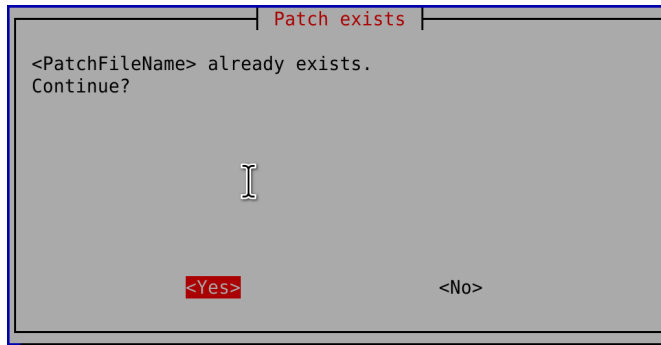


Abbildung 36.24.: Patch existiert bereits!

Der Befehl `dquilt new` mit dem Parameter des Patch-Namens erzeugt die erste "Patch-Datei".

Hierbei werden folgende Dateien erzeugt (Patch-Name exemplarisch):

```

debian/patches/series
 enthält den Namen des Patches DescriptionNoRegistration.patch
.pc/applied-patches
 enthält den Namen des Patches DescriptionNoRegistration.patch
.pc/DescriptionNoRegistration.patch/
.pc/.quilt_patches
 enthält debian/patches
.pc/.quilt_series
 enthält series
.pc/.version
 enthält 2

```

Wenn schon Patches existieren, ist darauf zu achten, dass diese in der Datei `debian/patches/series` in der korrekten Reihenfolge genannt werden.

```

281a <CreateNewPatch2 281a>≡ (280c)
 # Create a new patch file
 $dquilt new ${PatchFileName}

 # Patch
 FileToPatch
 <CreateNewPatch4 285b>

```

### 36.5.4. Datei zum Patchen auswählen

```

281b <PatchTasks4 281b>≡ (279c)
 2) FileToPatch # Add patch to patch file
 PatchRunNr=1
 $dquilt refresh;;
 <PatchTasks6 285c>

```

Wird ein neuer Patch erstellt oder soll ein weiterer Patch einer existierenden Patch-Datei hinzugefügt werden, ist die Quellcode-Datei auszuwählen, die verändert werden soll.

```

282 <FileToPatch 282>≡ (285a)
 function FileToPatch {
 # Called by CreateNewPatch PatchTasks and itself

 FileSelector ${GitPath} # Select the file to be patched
 File2Patch=${selected}

 echo "Patch ${PatchFileName} because of ${File2Patch}" >> ${log}

 # quilt add must get only the filename without
 # the path of the file to be patched
 File2pName=$(basename ${File2Patch})
 $dqilt add -P ${PatchFileName} ${File2pName}

 nano --linenumbers --mouse --softwrap ${File2Patch}

 $dqilt refresh ${PatchFileName}

 if whiptail --title "Patch another" \
 --yesno "Do you want to patch another file in ${PatchFileName}?" \
 --yes-button "Yes" --no-button "No" --defaultno 15 60
 then
 FileToPatch
 fi
 }

 <CreateNewPatch 280a>

```

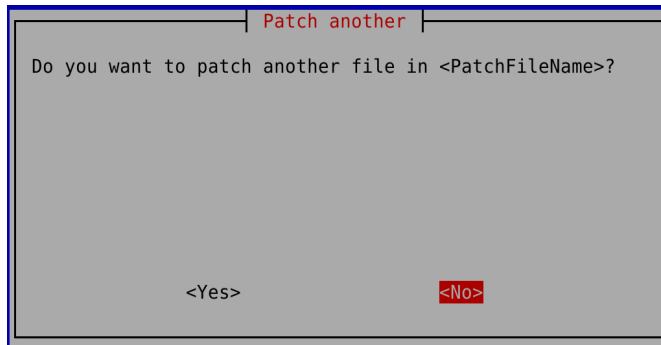


Abbildung 36.25.: Eine weitere Datei patchen?

Die Auswahl der zu korrigierenden Datei erfolgt in einem Dateiauswahldialog.

283 `<FileSelector 283>≡` (288)

```
function FileSelector {
 # Called by CreateNewPatch and itself
 # Dialog to select a file using whiptail

 StartPath=$1
 cd $StartPath
 txta=$(ls -a)

 i=0
 flist=''
 for element in ${txta[*]}
 do
 if [$element == '.']
 then
 i=$(expr $i + 1)
 continue
 fi
 flist=$flist' '$i' '${element}
 i=$(expr $i + 1)
 done

 sel=$(whiptail --title "Filepicker" \
 --menu "Select:" 15 60 6 $flist 3>&2 2>&1 1>&3)
```

`<FileSelector1 284>`

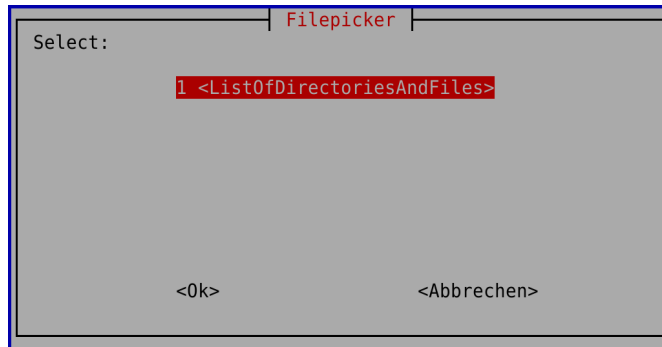


Abbildung 36.26.: Dateiauswahldialog

```

284 <FileSelector1 284>≡
 if [$? -ne 0]
 then
 return
 fi

 # Go back
 if [${txta[$sel]} = '..']
 then
 cd ..
 StartPath=$(pwd)
 selected=${StartPath}
 else
 selected=${StartPath}/${txta[$sel]}
 fi

 # The order of the following if-clauses is important
 if [-f ${selected}]
 then
 if ! whiptail --title "Your choice;" \
 --yesno "${selected}\nContinue?" --yes-button "Yes" \
 --no-button "No" 15 60
 then
 FileSelector ${StartPath}
 fi
 fi

```

(283)

<FileSelector2 285a>



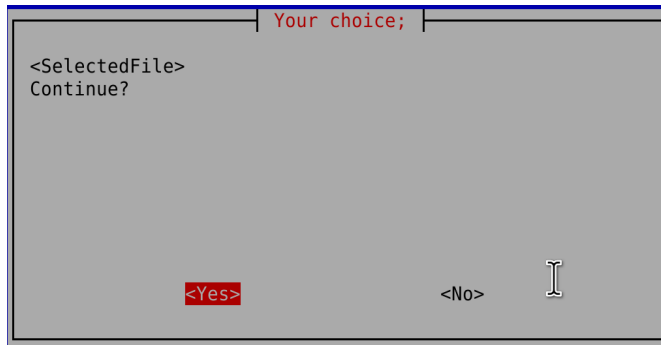


Abbildung 36.27.: Dateiauswahl bestätigen

```

285a <FileSelector2 285a>≡ (284)
 if [-d ${selected}]
 then
 FileSelector ${selected}
 fi
 }

```

<FileToPatch 282>

Sind in diesem Verzeichnis alle Dateien vorhanden, wird mit *dquilt add* mit dem Parameter der zu bearbeitenden Datei diese in *quilt* eingebunden.

Damit wird in *.pc/DescriptionNoRegistration.patch/* die genannte Datei hinzugefügt. Dort steht sie dann für den ersten Abgleich zur Verfügung.

```

285b <CreateNewPatch4 285b>≡ (281a)
 PatchHeader
 }

```

<CreateDquilt 276b>

### 36.5.5. Patch löschen

Mit der folgenden Auswahl werden bestehende Patches gelöscht. Dies ist notwendig, wenn ein bisheriger Patch nicht mehr benötigt wird, weil die Korrekturen inzwischen vom Upstream eingebaut wurden.

```

285c <PatchTasks6 285c>≡ (281b)
 3) DeletePatches;; # Delete patches
 <PatchTasks7 287a>

```

4. Mai 2025

Es kann aber auch sein, dass der bisherige Patch für die aktuelle Upstream-Version angepasst werden muss. Ein Weg hierzu ist, den alten Patch zu löschen und einen neuen zu erstellen.

```
286a <DeletePatches 286a>≡ (286b)
 function DeletePatches {
 # Called by PatchTasks and itself

 cd ${GitPath}
 DeletePatch
 PatchRunNr=1

 if whiptail --title "Another patch?" \
 --yesno "Do you want to delete another patch?" --yes-button "Yes" \
 --no-button "No" 15 60
 then
 DeletePatches
 fi
 }
```

<ApplyCheck 255>

Wird die Frage mit *No* beantwortet, springt das Programm zur Auswahl der Patchaufgaben (Kapitel 36.5, Seite 276) zurück.

```
286b <DeletePatch 286b>≡ (289b)
 function DeletePatch {
 # Called by DeletePatches

 ShowPatches "delete" # String will be found in ${1}

 if [-z "${PatchFileName}"]
 then
 PatchTasks
 fi

 less --LINE-NUMBERS ${GitPath}/debian/patches/${PatchFileName}

 if whiptail --title "Delete this patch?" \
 --yesno "Do you really want to delete ${PatchFileName}?" \
 --yes-button "Yes" --no-button "No" 15 60
 then
 $dquilt delete -r --backup ${PatchFileName}

 if whiptail --title "Delete backup file?" \
 --yesno "Do you want to delete the backup file, too?" \
 --yes-button "Yes" --no-button "No" 15 60
 then
 rm ${GitPath}/debian/patches/${PatchFileName}~
 fi
 fi
 }
```

<DeletePatches 286a>

### 36.5.6. *debian/patches/series* editieren

Ein Grund, die Datei *debian/patches/series* zu bearbeiten ist zu gewährleisten, dass in dieser Datei die Patches in der korrekten Reihenfolge aufgeführt werden.

```
287a <PatchTasks7 287a>≡ (285c)
 # Edit series
 4) nano --linenumbers --mouse --softwrap debian/patches/series;;
<PatchTasks8 287b>
```

### 36.5.7. Ausgangszustand wiederherstellen

Wenn ein Patch mit *quilt* erstellt, bearbeitet oder gelöscht wurde, werden beim Verlassen der Funktion die Patches aus dem Upstream-Code entfernt und dieser in den ursprünglichen Zustand zurückversetzt. Dies geschieht mit *quilt pop -a*. Dies bedeutet, dass alle angewandten Patches entfernt werden.

```
287b <PatchTasks8 287b>≡ (287a)
 5) if [${PatchRunNr} -eq 1]
 then
 # remove all patches and return the source
 # to its original state
 ${dqquilt} pop -a
 PatchRunNr=0
 fi
 # If debian/patches/series is empty,
 # delete directory debian/patches
 if ! [-s debian/patches/series]
 then
 rm debian/patches/series
 rmdir debian/patches
 fi
 return;;
 esac
 fi

 PatchTasks
}

<GitBranch2RecentBranch 306b>
```

4. Mai 2025

So ist es für Quilt möglich immer das Diff zwischen der Vorversion und der aktuellen Version zu erstellen. Danach erfolgt die Änderung. Die Informationen zur Registrierung werden entfernt. Mit *dquilt refresh* erzeugt man das Diff zur Dokumentation der Änderung. Mit *dquilt header -e* wird nun die Beschreibung der Änderung im \$EDITOR erstellt.

```
288 <PatchHeader 288>≡ (330a)
function PatchHeader {
 # Called by CreateNewPatch EditPatch

 PatchDescription=$(whiptail --title "Describe patch!" \
 --inputbox "Description:\n\n" \
 --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
 if [-z "${PatchDescription}"]
 then
 echo "Please insert the description of the patch!"
 read PatchDescription
 fi
 if whiptail --title "Describe patch!" --yesno "Forwarded:?" \
 --yes-button "Yes" --no-button "No" 15 60
 then
 PatchForwarded="Yes"
 else
 PatchForwarded="No"
 fi

 DEBValues
 if [-z ${Uploaders}]
 then
 PatchAuthor=${Maintainer}
 else
 PatchAuthor=${Uploaders}
 fi

 PatchUpdate=$(date +%Y'-'%m'-'%d)

 touch ${PatchFileName}.header
 echo "Description: ${PatchDescription} >> ${PatchFileName}.header"
 echo "Forwarded: ${PatchForwarded} >> ${PatchFileName}.header"
 echo "Author: ${PatchAuthor} >> ${PatchFileName}.header"
 echo "Last-Update: ${PatchUpdate} >> ${PatchFileName}.header"

 $dquilt header -a < ${PatchFileName}.header
 rm ${PatchFileName}.header
}

<FileSelector 283>
```

Hier ein Beispiel dazu:

```
Description: removed registration of license
Forwarded: No
Author: Mechtilde Stehmann <ooo@mechtilde.de>
Last-Update: 2014-05-18
```

### 36.5.8. Patch bearbeiten

```
289a <PatchTasks2 289a>≡ (279b)
 # Patches treatment
 case "$PTask" in
 0) ChangePatches;; # Edit patches
 <PatchTasks3 279c>
```

Zum Bearbeiten eines Patches wird die Funktion *EditPatch* aufgerufen.

```
289b <ChangePatches 289b>≡ (290)
 function ChangePatches {
 # Called by PatchTasks and itself

 cd ${GitPath}
 EditPatch

 if whiptail --title "Another patch?" \
 --yesno "Do you want to edit another patch?" --yes-button "Yes" \
 --no-button "No" 15 60
 then
 ChangePatches
 fi
 }

 <DeletePatch 286b>
```

4. Mai 2025

Diese ruft die Funktion *ShowPatches* mit dem Parameter *edit* auf.

```
290 <EditPatch 290>≡ (275b)
function EditPatch {
 # Called by ChangePatches

 ShowPatches "edit" # String will be found in ${1}

 if [-z "${PatchFileName}"]
 then
 return 1
 else
 PatchRunNr=1
 fi

 $dquilt pop ${PatchFileName}
 nano --linenumbers --mouse --softwrap debian/patches/${PatchFileName}
 $dquilt refresh

 if whiptail --title "New Patch Header" \
 --yesno "Do you want to create a new patch header?" --yes-button "Yes" \
 --no-button "No" 15 60
 then
 PatchHeader
 fi

 while $dquilt push
 do
 $dquilt refresh
 done
}

<ChangePatches 289b>
```

## 37. Bauen

Das eigentliche Bauen der Binärpakete erfolgt in einer *chroot*. Hierbei wird hauptsächlich geprüft, ob alle benötigten Buildabhängigkeiten in der Datei *debian/control* (Kapitel 35.4.6, Seite 235) aufgeführt sind und bereits als **Debian**-Pakete zur Verfügung stehen.

Damit wird gewährleistet, dass das Paket auch von den FTP-Mastern reproduzierbar ohne Zugriff auf weitere Netzressourcen gebaut werden kann.

### 37.1. debian/changelog

Vor dem eigentlichen Bauen wird die Datei *debian/changelog* zur Anpassung angezeigt.

```
291a <BuildNewRevision6 291a>≡ (253a)
 # Check debian/changelog
 # - includes creating changelog using gbp dch
 DisplayDebianChangelog
```

<MovingGbpConf 301a>

Zunächst wird im Skript geprüft, ob eine Datei *debian/changelog* bereits existiert. Wenn dies der Fall ist, wird diese Datei angezeigt und abgefragt, ob sie korrekt ist.

Andernfalls wird sie mit *gbp dch* erzeugt.

```
291b <DisplayDebianChangelog 291b>≡ (247b)
 function DisplayDebianChangelog {
 # Called by BuildNewRevision
```

```

 newChangelog=0
 if [-f debian/changelog]
 then
 less --LINE-NUMBERS debian/changelog
 if ! whiptail --title "Changelog ok?" --defaultno \
 --yesno "Is debian/changelog ok?" --yes-button "Yes" \
 --no-button "No" 15 60
 then
 newChangelog=1
 fi
 else
 newChangelog=1
 fi
 fi
```

<DisplayDebianChangelog1 292>

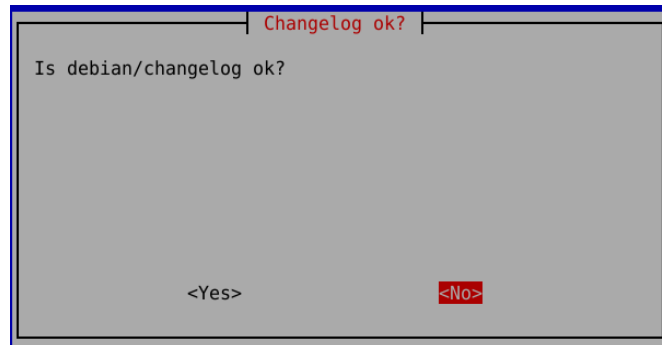


Abbildung 37.1.: Debian-Changelog OK?

Wird die Frage nach der Korrektheit des Changelogs bejaht, geht es mit einem eventuellen Verschieben der *gbp*-Konfigurationsdatei weiter (Kapitel 37.2, Seite 301).

Ist ein solches Verschieben der *gbp*-Konfigurationsdatei nicht erforderlich, werden die Parameter für *gbp buildpackage* festgelegt (Kapitel 37.3, Seite 302).

Andernfalls wird diese Datei im Editor angezeigt.

Zu achten ist besonders auf die Versions- und Revisionsbezeichnung in der ersten Zeile der Datei *debian/changelog*. Dies gilt hauptsächlich, wenn Dateien aus dem Upstream-Quellcode ausgeschlossen wurden (Kapitel 10.4.1.3, Seite 32).

Für einen Non-Maintainer-Upload (NMU) wird als erster Eintrag nach den Informationen zu der Version der Eintrag

```
* Non-maintainer upload
```

erwartet.

Existiert die Datei *debian/changelog* noch nicht, wird sie mit *gbp dch* erzeugt.

```
292 <DisplayDebianChangelog1 292>≡ (291b)
Check whether d/control exists without a comment in line 1
if [${newChangelog} -eq 1]
then
 if ! [-f debian/control]
 then
 DebianControlTemplate
 else
 set +e
 cat --number debian/control | grep '^1 ' | grep '#' > /dev/null
 if [$? -eq 0]
 then
 set -e
 DebianControlTemplate
 fi
 set -e
 fi
 # creating changelog using gbp dch

 AddVersionNumber
<DisplayDebianChangelog3 299>
```



### 37.1.1. Versionsbezeichnung einfügen

*gbp dch* benötigt die Angabe der aktuellen Versionsbezeichnung. Enthält die entsprechende Variable nicht diesen Wert, wird zur (manuellen) Eingabe dieser Versionsbezeichnung aufgefordert.

Dagegen kann *dch* die nächsthöhere Versionsbezeichnung mit *dch -increment* erstellen. Dies kann auch explizit mit *dch --newversion <Version>* erfolgen.<sup>1</sup>

Das Programmskript nutzt die zweite Möglichkeit.

```
293a <AddVersionNumber 293a>≡ (294b)
 function AddVersionNumber {
 # Called by DisplayDebianChangelog
 if [-z "${Version1}"]
 then
 RecentIdentifier
 fi
 }
```

<AddVersionNumber1 295a>

Zunächst wird geprüft, welche Versionsbezeichnung in der ersten Zeile der Datei *debian/changelog* eingetragen ist. Es wird abgefragt, ob eine dort gefundene Versionsbezeichnung übernommen werden soll. Existiert die Datei *debian/changelog* nicht, wird die Versionsnummer abgefragt (s. Seite 297).

```
293b <RecentIdentifier 293b>≡ (298)
 function RecentIdentifier {
 # Called by AddVersionNumber ForceOrig
 # Takes version number from debian/changelog, if it exists

 if [-f ${GitPath}/debian/changelog]
 then
 set +e
 firstLine=$(grep --line-number 'urgency=' ${GitPath}/debian/changelog | grep '^1:')
 set -e
 whiptail --title "First line:" \
 --msgbox "First line of debian/changelog;\n${firstLine}" 15 60
 recentId=$(echo ${firstLine} | sed --expression='s/^.*(//' | \
 sed --expression='s/).*//')
 <RecentIdentifier2 294a>
 }
```

---

<sup>1</sup>New Maintainer Guide, Kap. 8.1[11]

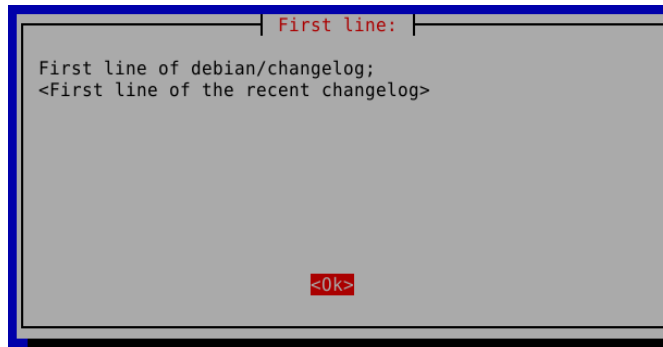


Abbildung 37.2.: Anzeige der ersten Zeile der Datei *debian/changelog*

294a  $\langle \text{RecentIdentifier2 } 294a \rangle \equiv$  (293b)  
`whiptail --title "Recent identifier" \  
--msgbox "Recent identifier is ${recentId}" 15 60`

$\langle \text{RecentIdentifier3 } 294b \rangle$

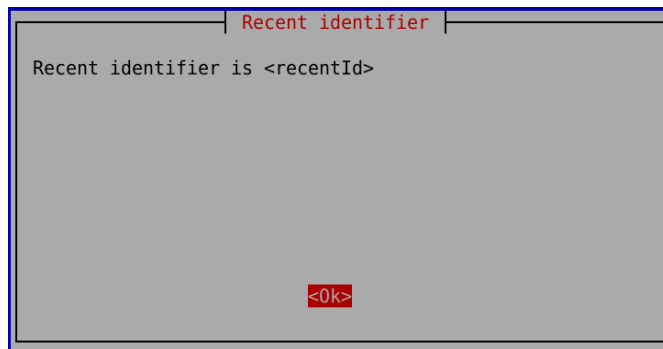


Abbildung 37.3.: Aktuelle Version

294b  $\langle \text{RecentIdentifier3 } 294b \rangle \equiv$  (294a)  
`if [ -n "${recentId}" ]  
then  
    Version1=${recentId}  
fi  
else  
    InsertIdentifier  
fi  
}  
  
 $\langle \text{AddVersionNumber } 293a \rangle$`

Danach geht es mit Kontrollfragen weiter.

```
295a <AddVersionNumber1 295a>≡ (293a)
 set +e
 revisionflag=$(echo ${Version1} | grep --count '\-[0-9]')
 set -e
 if [${revisionflag} -eq 0]
 then
 if ! whiptail --title "Identifier of the version:" \
 --defaultno --yesno "${Version1} contains no revision number.\n \
 Is it a native package?" --yes-button "Yes" --no-button "No" 15 60
```

<AddVersionNumber2 295b>

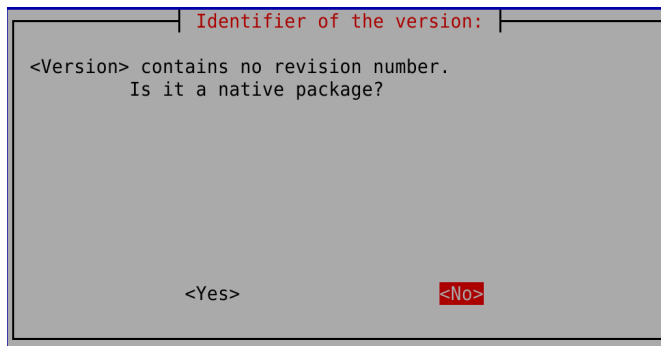


Abbildung 37.4.: Keine Revisionsnummer angegeben

```
295b <AddVersionNumber2 295b>≡ (295a)
 then
 InsertIdentifier
 if ! whiptail --title "Identifier of the version:" \
 --defaultno --yesno "Is ${Version1} the right identifier?" \
 --yes-button "Yes" --no-button "No" 15 60
```

<AddVersionNumber3 296>

4. Mai 2025

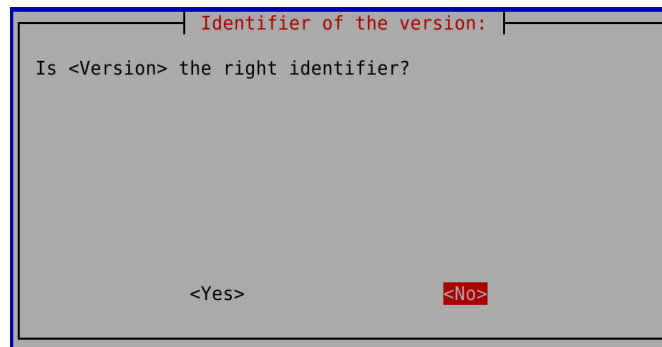


Abbildung 37.5.: Korrekte Versionsbezeichnung angeben?

```
296 <AddVersionNumber3 296>≡ (295b)
 then
 InsertIdentifier
 fi
 fi
 else
 if ! whiptail --title "Identifier of the version:" \
 --defaultno --yesno "Is ${Version1} the right identifier?" \
 --yes-button "Yes" --no-button "No" 15 60
 then
 InsertIdentifier
 fi
 fi
 echo "Message from AddVersionNumber: identifier=${Version1} >> ${log}"
}

<DebianJavabuildTemplate 247b>
```

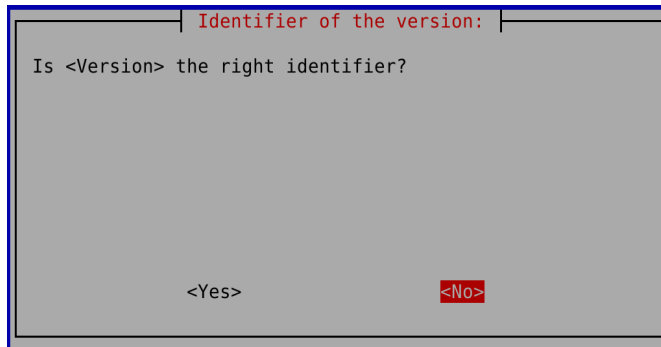


Abbildung 37.6.: Korrekte Versionsbezeichnung angeben?

In der Funktion *InsertIdentifier* wird zunächst der komplette Bezeichner des Paketes einschließlich der Versionsbezeichnung der Revision abgefragt.

297 `<InsertIdentifier 297>≡` (304b)

```
function InsertIdentifier {
 # Called by AddVersionNumber Recentidentifier
 RIdentifier=${Version1}
 set +e
 cat debian/source/format | grep "native" > /dev/null
 if [$? -eq 0]
 then
 Version1=$(whiptail --title "Identifier" \
 --inputbox "Recent identifier: ${RIdentifier}\n \
 Please insert the identifier of the package\n \
 (without revision version because it is a native package):" \
 --nocancel 15 60 3>&2 2>&1 1>&3)
```

`<InsertIdentifier1 298>`

4. Mai 2025

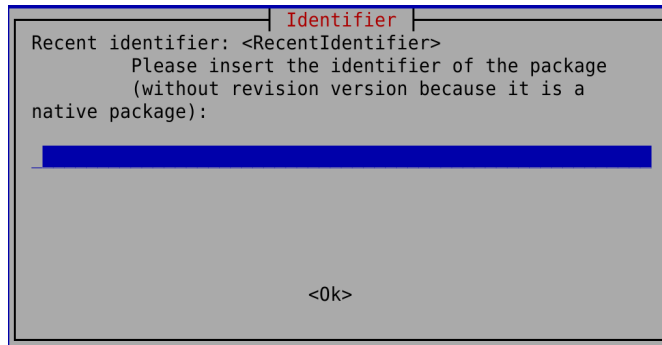


Abbildung 37.7.: Abfrage des Bezeichners eines nativen Paketes

```
298 <InsertIdentifier1 298>≡ (297)
 else
 Version1=$(whiptail --title "Identifier" \
 --inputbox "Recent identifier: ${RIdentifier}\n \
 Please insert the whole identifier of the package\n \
 (including revision version):" \
 --nocancel 15 60 3>&2 2>&1 1>&3)
 fi
 set -e
 }

 <RecentIdentifier 293b>
```

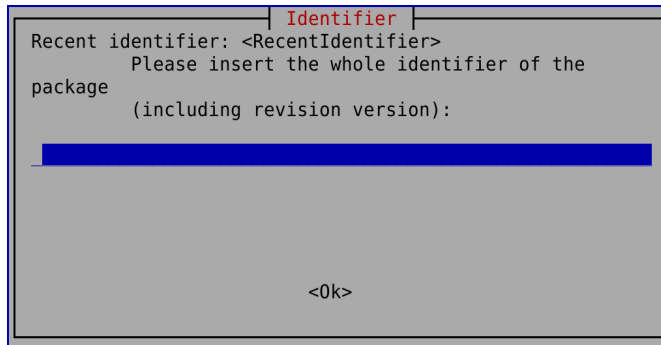


Abbildung 37.8.: Abfrage des Bezeichners

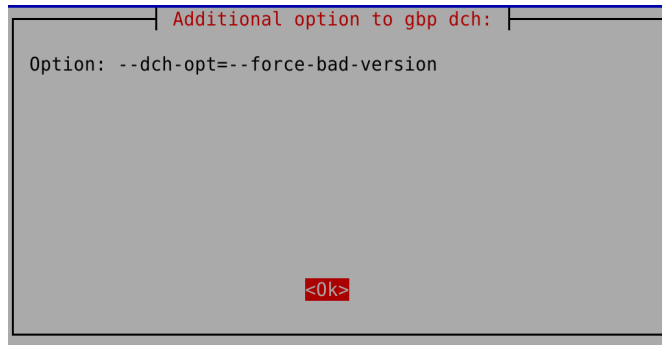
Anhand des Vorhandenseins einer Tilde wird geprüft, ob es sich um eine Schnappschuss-Version handelt. Ist dies der Fall, wird dem Befehl *gbp dch* auch die Option *--dch-opt=--force-bad-version* mitgegeben und dies angezeigt.

Dies bewirkt, dass das Programm *debchange* nicht stoppen wird, falls die neue Version kleiner als die aktuelle ist. Dies ist vor allem beim Rückportieren sinnvoll

```

299 <DisplayDebianChangelog3 299>≡ (292)
 set +e
 SnapshotFlag=$(echo ${Version1} | grep --count '~')
 if [${SnapshotFlag} -eq 0]
 then
 DchAdd=''
 else
 DchAdd=' --dch-opt=--force-bad-version'
 whiptail --title "Additional option to gbp dch:" \
 --msgbox "Option: ${DchAdd}" 15 60
 fi
<DisplayDebianChangelog4 300>

```

Abbildung 37.9.: Weitere Optionen für *dch*

Die Datei *debian/changelog* wird vom Programmskript mittels des Programms *gbp dch* erstellt. Immer wird *gbp dch* mit den Optionen *--verbose*, *--debian-branch=* und *--new-version=* aufgerufen.

Im Falle eines Fehlschlages von *gbp dch* wird eine Fehlermeldung ausgegeben. Andernfalls wird die Datei *debian/changelog* zum Editieren angezeigt.

```

300 <DisplayDebianChangelog4 300>≡ (299)
 gbp dch --verbose --debian-branch=${RecentBranch} \
 --new-version=${Version1}${DchAdd}
 if [$? -ne 0]
 then
 set -e
 FailureNotice
 fi
 set -e
 nano --linenumbers --mouse --softwrap debian/changelog
 fi
}

<CmeFix 251>

```



Für das Paket *tsync* ergab sich folgende Befehlszeile zum Bauen einer Fehlerkorrektur für Buster, das seinerzeit das stabile Release war.

```
gbp dch --verbose --debian-branch=debian/buster \
--new-version=1.16-1~deb10u1 --dch-opt=--force-bad-version
```

Zum Hintergrund ist folgendes auszuführen:

Mit `--dch-opt=<dch-option>` können dem Befehl `gbp dch` Optionen für `debchange` (`dch`) mitgegeben werden. Dabei ist zu beachten, dass `dbp dch` das Programm `dch` mehrfach aufruft und die Option jeweils bei allen Aufrufen übergibt. Daher sind an dieser Stelle nicht alle `dch`-Optionen sinnvoll. Außerdem kann es passieren, dass Optionen im Widerspruch stehen zu Optionen, die von `gbp dch` selbstständig übergeben werden.

Nach dem Schließen des Editors wird regelmäßig der Dialog zur Prüfung des Git-Zweiges angezeigt (Kapitel 37.3.2, Seite 305).

## 37.2. Verschieben der *gbp*-Konfigurationsdatei

Wenn für das zu bauende Paket eine spezielle Konfigurationsdatei für `git-buildpackage` verwandt werden soll (Kapitel 20.3, Seite 71), wird diese im Verzeichnis `debian/` mitveröffentlicht.

Wurde für `gbp import-orig` beim (erstmaligen) Herunterladen des Quellcodes eine solche Konfigurationsdatei erstellt (Kapitel 34.3.9, Seite 210), ist diese gegebenenfalls in das Verzeichnis `debian/` zu verschieben. Dies geschieht durch die Funktion `MovingGbpConfFile`.

```
301a <MovingGbpConf 301a>≡ (291a)
 MovingGbpConfFile
 <Preparations 302a>
```

Existiert eine Datei `gbp.conf` im Verzeichnis `.git/`, aber keine im `debian/`-Verzeichnis, so wird diese Datei nach `debian/` verschoben.

Sind in beiden Verzeichnissen entsprechende Dateien vorhanden, kann die zu veröffentlichende Datei ausgewählt werden.

```
301b <MovingGbpConfFile 301b>≡ (213a)
 function MovingGbpConfFile {
 # Called by BuildNewRevision

 # .git/gbp.conf exists, but not debian/gbp.conf
 # Move gbp.conf from .git/ to debian/
 if [-f ${GitPath}/.git/gbp.conf -a ! -f ${GitPath}/debian/gbp.conf]
 then
 mv -iv ${GitPath}/.git/gbp.conf ${GitPath}/debian
 fi
 # There is a gbp.conf in both directories
 if [-f ${GitPath}/.git/gbp.conf -a -f ${GitPath}/debian/gbp.conf]
 then
 TwoConfFilesFound
 fi
 }
```

<DebianBranch4Import 207>

Die Funktion `TwoConfFilesFound`, welche diese Auswahl ermöglicht, wird in (Kapitel 34.3.9, Seite 210) beschrieben.

### 37.3. Parameter für *gbp buildpackage* festlegen

Bevor das Bauen der Pakete beginnen kann, müssen zunächst Parameter für *gbp buildpackage* ermittelt und festgelegt werden.

```
302a <Preparations 302a>≡ (301a)
 # Preparations for gbp buildpackage
 AskDist # Ensure that RecentBranch has a value
 <Preparations1 305a>
```

#### 37.3.1. Git-Zweig und Distribution ermitteln

Damit im richtigen Git-Zweig und mit der richtigen Distribution gebaut wird, werden diese Parameter ermittelt und angezeigt. Sie können auch noch vom Benutzer angepasst werden.

Zunächst werden mit der Funktion *IdentifyBranches* die vorhandenen Git-Zweige ermittelt (Kapitel 32.5.2, Seite 154).

```
302b <AskDist 302b>≡ (307a)
 function AskDist {
 # Called by BuildNewRevision PrepareUploading LastQuestionsBeforeBuild

 IdentifyBranches
 ba=($bl)
 set +e
 for element in ${ba[*]}
 do
 # rb=$(echo ${element} | grep --count '^x_')
 # if [$rb -ge 1]
 if echo ${element} | grep --quiet '^x_'
 then
 CurrentBranch=$(echo ${element} | sed --expression='s/^x_//')
 fi
 done
 set -e

 <AskDist0 302c>
```

Zunächst wird mit *-z* geprüft, ob die Variable *RecentBranch* leer ist. In diesem Fall wird die Funktion *GitBranch2RecentBranch* (Kapitel 37.3.3, Seite 306) aufgerufen.

Andernfalls wird geprüft, ob der Name des aktuellen Zweiges dem Wert der Variablen *RecentBranch* entspricht. Sollte dies nicht der Fall sein, erfolgt ein Hinweis und der Nutzer kann einen der beiden Zweige auswählen.

```
302c <AskDist0 302c>≡ (302b)
 if [-z "${RecentBranch}"]
 then
 GitBranch2RecentBranch
 else
 if ["${RecentBranch}" != "${CurrentBranch}"]
 then
 Msg="Branch according to git: "${CurrentBranch}","\n \
 branch according to "${ConfigPath}${OrigName}": "${RecentBranch}
 whiptail --title "There is something wrong!" --msgbox "${Msg}" 15 60
 <AskDist1 303>
```

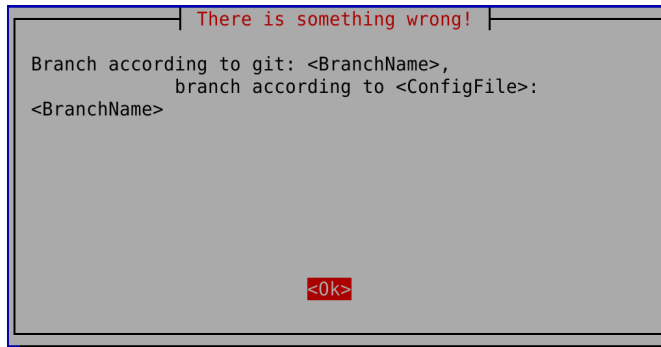


Abbildung 37.10.: Da läuft was falsch!

303 &lt;AskDist1 303&gt;≡

```

WishedBranch=$(whiptail --title "Choose branch:" \
--radiolist "Which branch do you want to work with? " \
--cancel-button "Cancel" 15 60 2 \
"0" "${RecentBranch}" off \
"1" "${CurrentBranch}" off 3>&2 2>&1 1>&3)

```

&lt;AskDist2 304a&gt;

(302c)

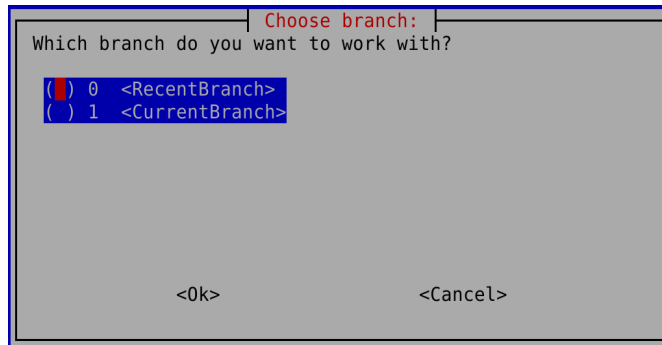


Abbildung 37.11.: Auswahl des Branches

304a  $\langle AskDist2 \ 304a \rangle \equiv$  (303)

```

if [${WishedBranch} -eq 0]
then
 git checkout ${RecentBranch}
else
 GitBranch2RecentBranch
fi
fi

echo "Notice from AskDist: The branch is "${RecentBranch} >> ${log}
set +e
va=$(grep --count ${RecentBranch}_Dist ${ConfigPath}${OrigName})
if [$va -eq 1]
then
 bName=${RecentBranch}
 set -e
 Search4Dist
 RecentBranchD=${va}
elif [$va -gt 1]
then
 nano ${ConfigPath}${OrigName}
 AskDist

```

$\langle AskDist5 \ 304b \rangle$

304b  $\langle AskDist5 \ 304b \rangle \equiv$  (304a)

```

else
 set -e
 Distro4Branch
fi
set -e

if [-z "${RecentBranchD}"]
then
 RecentBranchD="sid"
fi
echo "Notice from AskDist: The distribution is "${RecentBranchD} >> ${log}
}

```

$\langle InsertIdentifijer \ 297 \rangle$

### 37.3.2. Git-Zweig und Distribution prüfen

Der Git-Zweig wird angezeigt, damit der Nutzer prüfen kann, ob es der richtige ist.

```
305a <Preparations1 305a>≡ (302a)
 echo "Notice from BuildNewRevision: Branch is "${RecentBranch} >> ${log}
 whiptail --title "Please check!" \
 --yesno "The git branch is ${RecentBranch}" --yes-button "Yes" \
 --no-button "No" 15 60
 rbq=$?
 <Preparations2 305b>
```

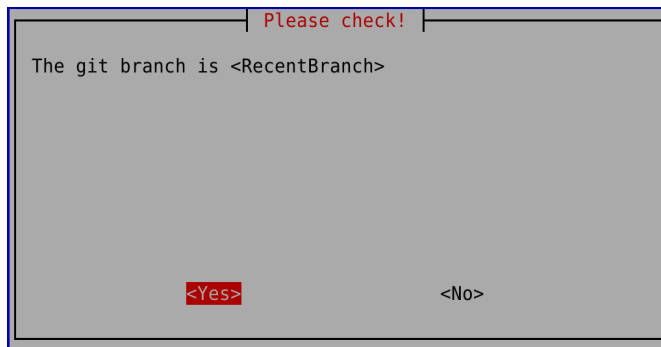


Abbildung 37.12.: Anzeige des Git-Zweiges

Wird die Frage, ob der angezeigte Git-Zweig der richtige ist, verneint, kann der zutreffende Zweig ausgewählt werden (Kapitel 33.4, Seite 169).

```
305b <Preparations2 305b>≡ (305a)
 if [$rbq -ne 0]
 then
 SelectBranch
 fi
 <Preparations3 305c>
```

Wird die Frage, ob die angezeigte Distribution die richtige ist, verneint, kann die zutreffende Distribution ausgewählt werden (Kapitel 32.5.3, Seite 154).

```
305c <Preparations3 305c>≡ (305b)
 if ! whiptail --title "Please check!" \
 --yesno "The distribution is ${RecentBranchD}" --yes-button "Yes" \
 --no-button "No" 15 60
 <Preparation4 306a>
```

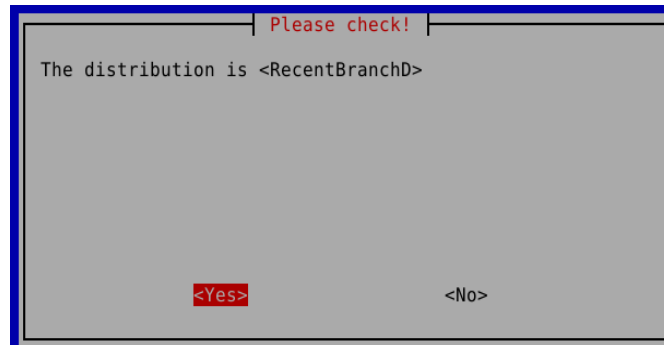


Abbildung 37.13.: Release-Branch der Distribution

```

306a <Preparation4 306a>≡ (305c)
 then
 RecentBranchD=""
 Distro4Branch
 fi

 echo "Notice from BuildNewRevision: Distribution is "${RecentBranchD} >> ${log}

 SBuildOrPBuilder 0

 <BuildNewRevision8 321>

```

Zur Auswahl des Build-Systems wird die Funktion *SBuildOrPBuilder* aufgerufen (Kapitel 37.3.5, Seite 307).

### 37.3.3. Git-Zweig anpassen

Zunächst wird im Wert der Variablen *CurrentBranch* (welcher der Name des aktuellen Zweiges ist) ein "/" durch "\/" ersetzt, also ein Schrägstrich maskiert.

Dann wird der so veränderte Wert der Variablen als *RecentBranch* in die Konfigurationsdatei eingefügt.

Schließlich wird dieser Wert der Variablen *RecentBranch* zugewiesen.

```

306b <GitBranch2RecentBranch 306b>≡ (287b)
 function GitBranch2RecentBranch {
 # Called by AskDist

 bName1=$(echo ${CurrentBranch} | sed --expression='s/\//\\/g')
 sed --in-place --expression="s/RecentBranch=.*\/RecentBranch=${bName1}/g" \
 ${ConfigPath}${OrigName}
 RecentBranch=${CurrentBranch}
 }

 <Search4Dist 307a>

```

### 37.3.4. Distribution ermitteln

```
307a <Search4Dist 307a>≡ (306b)
function Search4Dist {
 # Called by AskDist ParseConfig
 set +e
 va=$(grep ${bName}_Dist ${ConfigPath}${OrigName})
 bName1=$(echo ${bName} | sed --expression='s/\\/\\\\/g')
 va=$(echo $va | sed --expression="s/# ${bName1}_Dist=//g")
 va=$(echo $va | sed --expression='s/"//g')
 set -e
}

<AskDist 302b>
```

### 37.3.5. Auswahl des Build-Systems

Nun kann noch die Auswahl getroffen werden, ob mit *pbuilder* oder *sbuild* gebaut werden soll.

```
307b <SBuildOrPBuilder 307b>≡ (320)
function SBuildOrPBuilder {
 # Called by BuildNewRevision TaskSelect PrepareUploading Import4Sponsoring
 # Flag for tagging and signing
 tsf=$1

 Builder=$(whiptail --title "Which builder do you want to use?" \
 --radiolist "Which builder do you want to use? " 15 60 6 \
 "0" "PBuilder" off \
 "1" "SBuild" on \
 --cancel-button "Exit" 3>&2 2>&1 1>&3)
 <SBuildOrPbuilder1 308a>
```

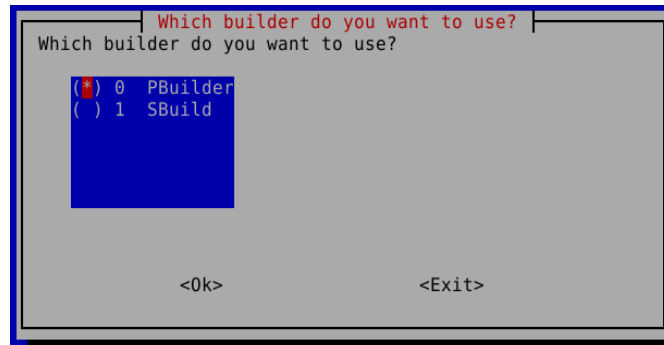


Abbildung 37.14.: Auswahl des Build-Systems

Voreingestellt ist die Auswahl von *sbuild*.

```
308a <SBUILDOrPbuilder1 308a>≡ (307b)
 if [-z "${Builder}"]
 then
 exit
 fi
 if ["${Builder}" -eq 1]
 then
 echo "Using SBuild." >>${log}
 UsingSBuild
 else
 echo "Using PBuilder." >>${log}
 UsingPBuilder
 fi
}
```

<BuildNewRevision 226>

Wird *SBuild* ausgewählt, ruft das Programmskript die Funktion *UsingSBuild* auf. Wird *PBuilder* ausgewählt, wird die Funktion *UsingPBuilder* aufgerufen.

Eine Beschreibung des *PBuilders* finden Sie im Kapitel 37.5 (Seite 313).

Was *sbuild* macht, erfährt man in Kapitel 37.4 (Seite 310).

Zuvor erhält der Nutzer jedoch die Gelegenheit vor dem Bau des Paketes die Parameter letztmalig zu prüfen. Außerdem wird ihm eine letzte Ausstiegsmöglichkeit gewährt (Kapitel 37.3.7, Seite 309).

### 37.3.6. Überprüfung der Parameter

Bevor der Paketbau endgültig beginnt, werden die Parameter letztmalig zur Prüfung angezeigt.

```
308b <LastQuestionsBeforeBuild 308b>≡ (257a)
 function LastQuestionsBeforeBuild {
 # Called by UsingSBuild UsingPBuilder

 if ! whiptail --title "Please check!" \
 --yesno "The release you want to build for in ${BuildEnv} is ${RecentBranchD}" \
 --yes-button "Yes" --no-button "No" 15 60
 then
 AskDist
 fi
 }

 <LastQuestionsBeforeBuild1 309a>
```



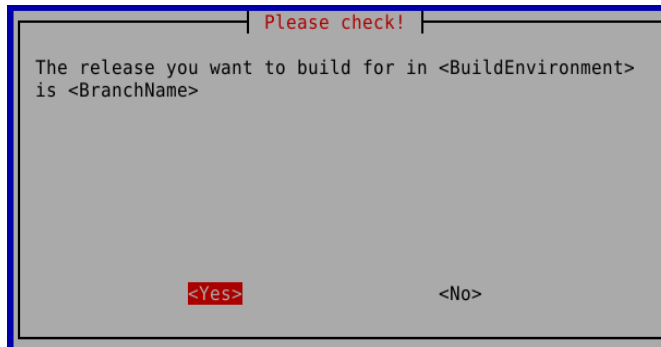


Abbildung 37.15.: Parameterüberprüfung

### 37.3.7. Letzte Ausstiegsmöglichkeit

Vor dem Paketbau wird schließlich eine letzte Gelegenheit zum Ausstieg gegeben.

```
309a <LastQuestionsBeforeBuild1 309a>≡ (308b)
 whiptail --title "Last opportunity to exit before building" \
 --yesno "Do you want to start the build process?" --yes-button "Yes" \
 --no-button "Exit" 15 60
 <LastQuestionsBeforeBuild2 309b>
```

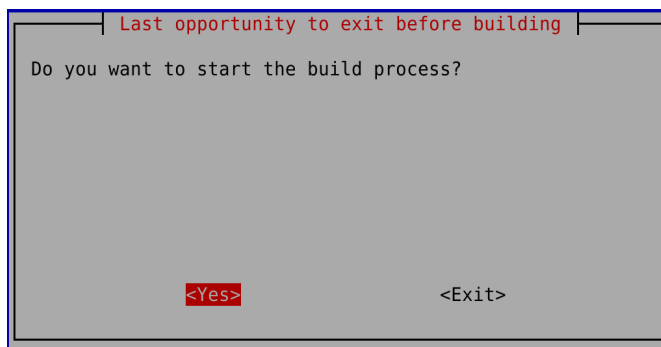


Abbildung 37.16.: Bau des Paketes starten

```
309b <LastQuestionsBeforeBuild2 309b>≡ (309a)
 if [$? -ne 0]
 then
 whiptail --title "Bye" --msgbox "Exit" 15 60
 <LastQuestionsBeforeBuild3 310a>
```

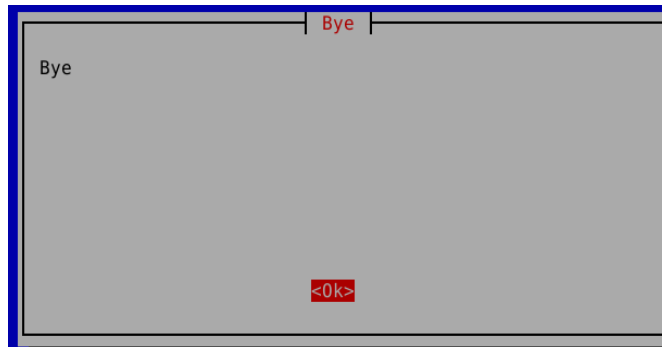


Abbildung 37.17.: Beenden

310a  $\langle$ *LastQuestionsBeforeBuild3* 310a $\rangle$ ≡ (309b)

```

exit
else
 if ["${BuildEnv}" = "pbuilder"]
 then
 whiptail --title "Start building" \
 --msgbox "${BuildEnv} will be updated first\n \
 This need sudo and/or root rights" 15 60
 fi
fi
}

```

$\langle$ *UsingSBuild* 310b $\rangle$

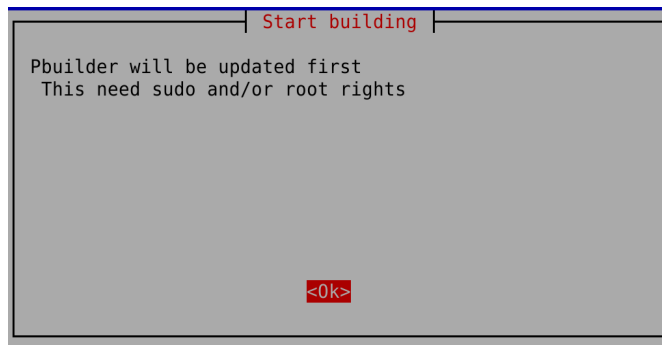


Abbildung 37.18.: Information zur Aktualisierung der Build-Umgebung

### 37.4. Was macht *sbuild*?

*Sbuild* wird im offiziellen *buildd*-Netzwerk verwendet, um Binär- und Quellpakete für alle unterstützten Architekturen zu erstellen[34].

Es wird dafür jeweils eine eigene Buildumgebung erstellt.

310b  $\langle$ *UsingSBuild* 310b $\rangle$ ≡ (310a)

```

function UsingSBuild {
 # Called by BuildNewRevision

 BuildEnv="sbuild"
 \langle UsingSBuild1 311 \rangle
}

```

Zunächst erfolgt der Aufruf der Funktion *LastQuestionsBeforeBuild* zur Festlegung der Parameter *release* und des dazugehörigen Git-Zweiges für *gbp buildpackage* (Kapitel 37.3.7, Seite 309).

Dann besteht die Gelegenheit, die Datei *~/.sbuilderc* zu prüfen und gegebenenfalls anzupassen.

```

311 <UsingSBuild1 311>≡ (310b)
 LastQuestionsBeforeBuild

 if whiptail --title ".sbuilderc." \
 --yesno "Do you want to check (and edit) ~/.sbuilderc?" \
 --defaultno --yes-button "Yes" --no-button "No" 15 60
 then
 nano ~/.sbuilderc
 fi
 <UsingSBuild2 312>

```

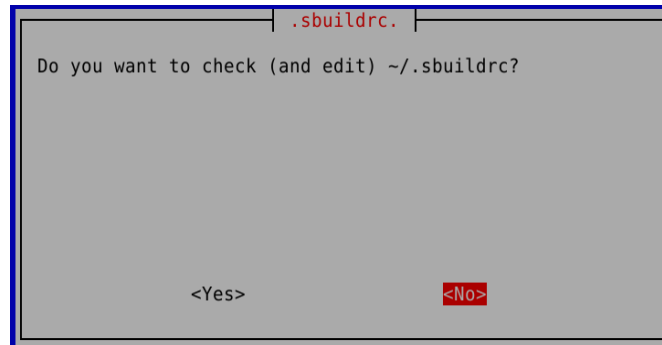


Abbildung 37.19.: .sbuilddrc prüfen

Wenn die Datei `~/.sbuilddrc` leer ist, ist diese mit dem beschriebenen Inhalt zu füllen (s. Kapitel 19.4, Seite 68)

Abhängig davon, zu welchem Zwecke der Paketbau erfolgen soll, wird `gbp buildpackage` mit einer unterschiedlichen Parameterkonfiguration aufgerufen.

312 `<UsingSbuild2 312>≡`

(311)

```

Check whether chroot directory exists

if ! [-d ~/.cache/sbuild]
then
 mkdir -p ~/.cache/sbuild
fi

Create/Update tarballed chroot

if [$tsf -eq 0]
Called by TaskSelect via SBuildOrPbuilder 0
then
 ChrootName='UNRELEASED'
else
 ChrootName=${RecentBranchD}
fi

mmdebstrap --variant=buildd ${RecentBranchD} \
~/.cache/sbuild/${ChrootName}-amd64.tar.xz

set +e
if [$tsf -eq 0]
Called by TaskSelect via SBuildOrPbuilder 0
then
 gbp buildpackage --git-builder=sbuild --no-clean-source \
--dist=UNRELEASED --build-failed-commands=%SBUILD_SHELL \
--git-debian-branch=${RecentBranch} \
--git-dist=${RecentBranchD} --git-ignore-new
 # Setting flag for success
 gbpq=$?
elif [$tsf -eq 1]
Called by PrepareUploading via SBuildOrPBuilder 1
then
 gbp buildpackage --git-builder=sbuild --no-clean-source \
--dist=${RecentBranchD} --git-debian-branch=${RecentBranch} \
--git-dist=${RecentBranchD} --git-verbose --git-tag \

```

```

--git-sign-tags
Setting flag for success
gbpq=$?
else
Called by PrepareUploading via SBuildOrPBuilder 2
gbp buildpackage --git-builder=sbuild --no-clean-source\
--dist=${RecentBranchD} --git-debian-branch=${RecentBranch} \
--git-dist=${RecentBranchD}
Setting flag for success
gbpq=$?
fi
set -e
}
<UsingPBuilder 315>

```

## 37.5. In der *Pbuilder-Chroot* bauen

Mit *PBuilder* wird eine Umgebung geschaffen, in der Debian-Pakete erstellt werden können (s. a. Kapitel 19.3, Seite 60).

Die Pakete werden in einer *chroot* gebaut, die von *pbuilder* bereitgestellt wird. Zur besseren Kontrolle des Prozesses können an vordefinierten Stellen sogenannte *hooks* eingebaut werden (Kapitel 19.3.3, Seite 64).

Da *git pbuilder* zur Aktualisierung der Basisinstallation *root*-Rechte benötigt, erfolgt eine Passwortabfrage.

### 37.5.1. *base.cow* erstellen

Ist die erforderliche *base-cow* noch nicht vorhanden, wird sie mit *git-pbuilder create* angelegt. Damit kann auch eine *chroot* für eine abweichende Architektur wie zum Beispiel *i386* (*32-bit*) auf einem *64-bit*-Rechner angelegt werden. Dazu wird eine *base-sid-i386.cow* mit

```
DIST=sid ARCH=i386 git-pbuilder create
```

erstellt.

```

313 <CreateNewCow 313>≡ (154a)
function CreateNewCow {
Called by Distro4Branch UsingPBuilder

bDist="sid"
set +e
bDist=$(whiptail --title "Create new cow for ${RecentBranch}" \
--inputbox "Debian distribution\n \
for this branch ${RecentBranch}:" \
--cancel-button "Use Sid" 15 60 3>&2 2>&1 1>&3)
<CreateNewCow1 314>

```

4. Mai 2025

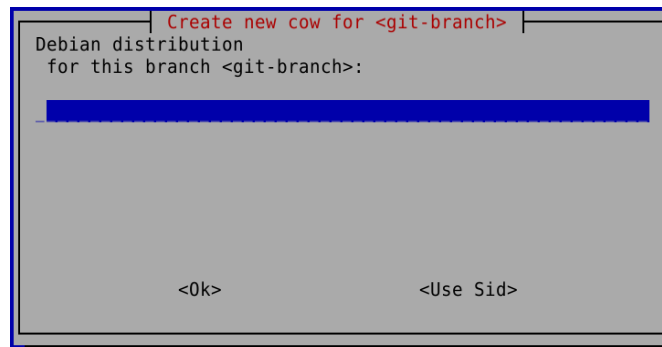


Abbildung 37.20.: Auswahl der zu erstellenden Cow.

```
314 <CreateNewCow1 314>≡ (313)
 if [-z "${bDist}"]
 then
 bDist="sid"
 fi
 # You must be root to create a cow
 echo -e "\nPlease enter Password for creating pbuilder cow.\n"
 sudo DIST=${bDist} git-pbuilder create
 echo "Cow for "${bDist}" created." >> ${log}
 set -e
}

<Distro4Branch 154b>
```

### 37.5.2. git-pbuilder update

Vor dem Start von *gbp buildpackage* sind Vorkehrungen zu treffen, damit die Einrichtung in der *chroot* aktuell ist. Dies geschieht mittels *git-pbuilder update*. Zuvor wird noch geprüft, ob die notwendige *base-cow* (des entsprechenden Branches/Releases) vorhanden ist.

Während die *cow*-Verzeichnisse für die übrigen Veröffentlichungen die Release-Bezeichnungen im Namen tragen, heißt das entsprechende Verzeichnis für *sid* schlicht *base.cow*.

Daher wird anhand zweier Bedingungen geprüft, ob das entsprechende Verzeichnis vorhanden ist.

Fehlt das entsprechende Verzeichnis wird es mittels der Funktion *CreateNewCow* (Kapitel 37.5.1, Seite 313) erstellt.

315 `<UsingPBuilder 315>≡` (312)

```
function UsingPBuilder {
 # Called by BuildNewRevision

 BuildEnv="pbuilder"
 LastQuestionsBeforeBuild

 # Building package using git-pbuilder and gbp buildpackage

 # check, whether cow exists
 # if exists update cow
 # else create cow
 if [-d /var/cache/pbuilder/base- $\{\text{RecentBranchD}\}$.cow]
 then
 echo -e "\nPlease enter Password for updating pbuilder cow.\n"
 DIST= $\{\text{RecentBranchD}\}$ git-pbuilder update
 echo "Notice from BuildNewRevision: Pbuilder was updated." >> $\{\text{log}\}$
 elif [-d /var/cache/pbuilder/base.cow -a $\{\text{RecentBranchD}\}$ = "sid"]
 then
 echo -e "\nPlease enter Password for updating pbuilder cow.\n"
 DIST= $\{\text{RecentBranchD}\}$ git-pbuilder update
 echo "Notice from BuildNewRevision: Pbuilder was updated." >> $\{\text{log}\}$
 else
 CreateNewCow
 fi

 if [$\{\text{OptFlag}\}$ -ne 1]
 then
 ForceOrig
 fi
}
```

`<UsingPBuilder3 316a>`

Für die Anzeige der bereits gesetzten Optionen für *gbp buildpackage* in der Funktion *MoreOptions* werden diese mit der Variablen *normalOpts* übergeben.

```
316a <UsingPBuilder3 316a>≡ (315)
 if [$tsf -eq 0]
 then
 normalOpts="--git-debian-branch=${RecentBranch}" \
 --git-dist=${RecentBranchD}" --git-verbose \
 --git-ignore-new${pbuilderOpt}
 elif [$tsf -eq 1]
 then
 normalOpts="--git-debian-branch=${RecentBranch}" \
 --git-dist=${RecentBranchD}" --git-verbose --git-tag \
 --git-sign-tags${pbuilderOpt}
 else
 normalOpts="--git-debian-branch=${RecentBranch}" \
 --git-dist=${RecentBranchD}" --git-verbose${pbuilderOpt}
 fi
 MoreOptions
fi
<UsingPBuilder4 320>
```

### 37.5.3. Aufnahme des \*.orig-Archives in \*.changes

Grundsätzlich wird das \*.orig-Archiv (*\*.orig.tar.gz* oder *\*.orig.tar.xz*) nur dann in die \*.changes-Datei eingebunden und damit auch hochgeladen, wenn die Revisionsnummer 1 nicht übersteigt.

Zunächst wird also die Versionsbezeichnung durch Aufruf der Funktion *RecentIdentifier* (Kapitel 37.1.1, Seite 293) ermittelt und angezeigt.

```
316b <ForceOrig 316b>≡ (231a)
function ForceOrig {
 # Called by BuildNewRevision PrepareUploading
 OptFlag=1
 if [-z "${Version1}"]
 then
 RecentIdentifier
 fi
 whiptail --title "Version" --msgbox "Version: ${Version1}" 15 60
}
<ForceOrig2 317a>
```



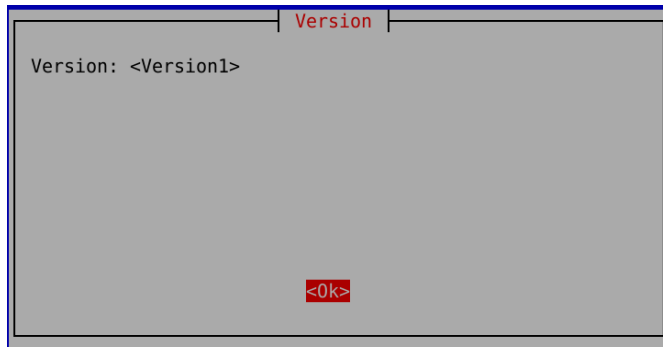


Abbildung 37.21.: Anzeige der Version mit Revisionsnummer

Es wird geprüft, ob es sich um ein *natives* Paket (Kapitel 16.1, Seite 51) handelt.

```
317a <ForceOrig2 317a>≡ (316b)
 set +e
 cat debian/source/format | grep "native" > /dev/null
 if [$? -ne 0]
 <ForceOrig3 317c>
```

Handelt es sich um ein natives Paket, so enthält die Versionsbezeichnung keine Revisionsbezeichnung.

```
317b <ForceOrig7 317b>≡ (318b)
 else
 pbuilderOpt=" --git-pbuilder"
 fi
 set -e
 }

 <MoreOptions 319a>
```

Bei nicht-nativen Debian-Paketen wird die Revisionsnummer extrahiert und angezeigt.

```
317c <ForceOrig3 317c>≡ (317a)
 then
 RevNr=$(echo ${Version1} | sed --expression='s/[^0-9]/#/g' | \
 sed --expression='s/^.*/#/')
 whiptail --title "Revision number" \
 --msgbox "The number of the revision is ${RevNr}." 15 60
 pbuilderOpt=" --git-builder=git-pbuilder \
 --git-pbuilder-options='--source-only-changes'"

 <ForceOrig5 318a>
```

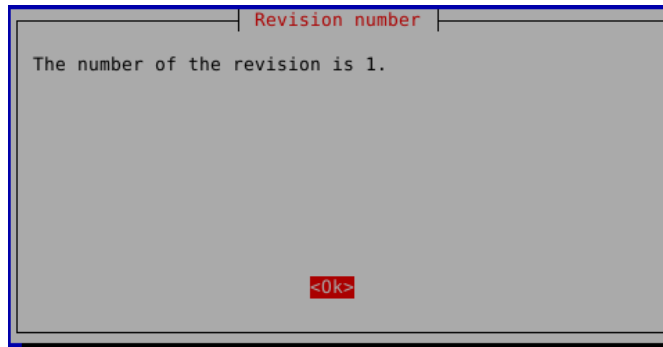


Abbildung 37.22.: Anzeige der Revisionsnummer

Man kann jedoch bei einer Revisionsnummer größer als 1 erzwingen, dass das \*.orig-Archiv in die \*.changes-Datei eingebunden und damit auch hochgeladen wird, indem *git-pbuilder* die Option *-sa* mitgegeben wird.

Dies ist dann sinnvoll, wenn das \*orig-Archiv bislang noch nicht hochgeladen wurde oder in der *New Queue* erneut bereitgestellt werden muss.

Bei der Verwendung von *gbp buildpackage*, wie im Skript, lautet die entsprechende Option *-git-builder=git-pbuilder -sa*.

```
318a <ForceOrig5 318a>≡ (317c)
 if [${RevNr} -gt 1]
 then
 if whiptail --title "orig in changes file" \
 --yesno "Do you want to insert the orig archive into the changes file?\n \
 That makes sense, if the orig archive has not been uploaded before." \
 --defaultno --yes-button "Yes" --no-button "No" 15 60
 <ForceOrig6 318b>
```

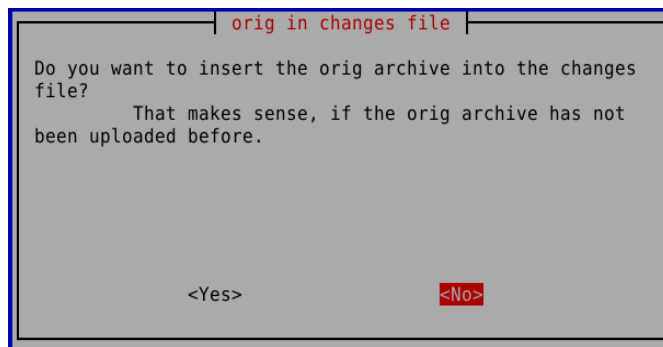


Abbildung 37.23.: Soll der Upstream-Tarball auch hochgeladen werden

Die Option *-sa* bedeutet, dass laut *dpkg-buildpackage -help* die Quelle immer *Orig* enthält.

```
318b <ForceOrig6 318b>≡ (318a)
 then
 pbuilderOpt=" --git-builder=git-pbuilder -sa"
 fi
 fi
 <ForceOrig7 317b>
```

Nun wird in der folgenden Funktion die Möglichkeit eröffnet, weitere Optionen für den *pbuilder* in *gpb buildpackage* mitzugeben. Zuvor werden die bisherigen Optionen angezeigt.

```
319a <MoreOptions 319a>≡ (317b)
function MoreOptions {
 # Called by UsingPBuilder PrepareUploading
 # Adds options to specify pbuilder in gpb buildpackage
 moreOpts=''
 intText="The options for gpb buildpackage are:\n"
 intText=${intText}${normalOpts}
 if whiptail --title "Options for gpb buildpackage" \
 --yesno "${intText}\nDo you want to add some more?" --yes-button "Yes" \
 --no-button "No" --defaultno 15 60
 <MoreOptions2 319b>
```

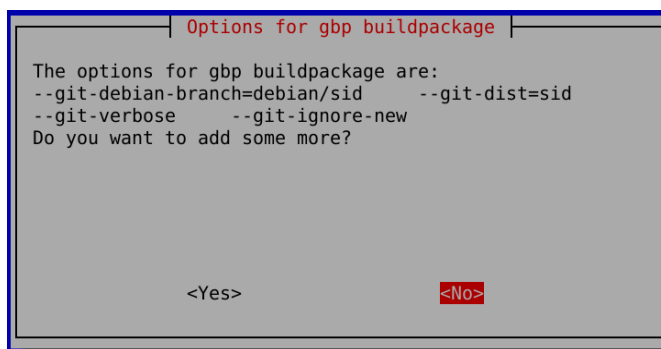


Abbildung 37.24.: Anzeige der Optionen von *gpb buildpackage*

```
319b <MoreOptions2 319b>≡ (319a)
 then
 moreOpts=$(whiptail --title "Options for gpb buildpackage" \
 --inputbox "${intText}\nPlease insert options to be added:" \
 --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
 moreOpts=" ${moreOpts}"
 fi
}

<PatchesTreatment 253b>
```

Die Informationen zu *pbuilderOpt* wurden aus verschiedenen Manpages zusammengetragen.

In der Manpage zu *gbp buildpackage* erhält man die Information, dass mit *-git-pbuilder-options= PUILDER\_OPTIONS* weitere Optionen des *pbuilders* hinzugefügt werden können. Welche Optionen dies sind, findet man in der Manpage von *pbuilder*

### 37.5.4. Bauen mit *gbp buildpackage*

Nach den Vorbereitungen erfolgt nun das Bauen des jeweiligen Paketes mit *gbp buildpackage*. Misslingt dies, beendet sich das Programm. Andernfalls geht es weiter zur Überprüfung der gebauten Pakete (Kapitel 40, Seite 327). Good luck!

Hier beginnt nun mit dem Herunterladen der Build-Abhängigkeiten das eigentliche Bauen des Debian-Paketes.

```

320 <UsingPBuilder4 320>≡ (316a)
 echo "== Options for gbp buildpackage ==" >> ${log}
 echo -e "RecentBranch contains ${RecentBranch}" >> ${log}
 echo -e "RecentBranchD contains ${RecentBranchD}" >> ${log}
 echo -e "PBuilderOpt contains ${pbuilderOpt}" >> ${log}
 echo -e "MoreOpts contains ${moreOpt} \n" >> ${log}
 echo "Starting gbp buildpackage" >> ${log}
 set +e

 if [$tsf -eq 0]
 then
 gbp buildpackage --git-debian-branch=${RecentBranch} \
 --git-dist=${RecentBranchD} --git-verbose \
 --git-ignore-new${pbuilderOpt}${moreOpts}
 # Setting flag for success
 gbpq=$?
 elif [$tsf -eq 1]
 then
 gbp buildpackage --git-debian-branch=${RecentBranch} \
 --git-dist=${RecentBranchD} --git-verbose --git-tag \
 --git-sign-tags${pbuilderOpt}${moreOpts}
 # Setting flag for success
 gbpq=$?
 else
 gbp buildpackage --git-debian-branch=${RecentBranch} \
 --git-dist=${RecentBranchD} --git-verbose${pbuilderOpt}${moreOpts}
 # Setting flag for success
 gbpq=$?
 fi
 set -e
}

<SBuildOrPBuilder 307b>

```

```

321 <BuildNewRevision8 321>≡ (306a)
 if [$gbpq -eq 0]
 then
 echo -e "The package ${SourceName} was built with gbp buildpackage\n \
 without creating and signing tags." >> ${log}
 else
 whiptail --title "Build failed!" \
 --msgbox "Gbp buildpackage failed!" 15 60
 echo
 echo "-- Gbp buildpackage failed! --"
 echo
 echo "Please fix the problem in another terminal!"
 echo "After fixing, press RETURN to continue."
 read a
 if ["${BuildEnv}" = "sbuild"]
 then
 UsingSBuild
 else
 UsingPBuilder
 fi
 fi
 Task=5 # Go to RunningTests
 }

```

<RunningLintian 330b>

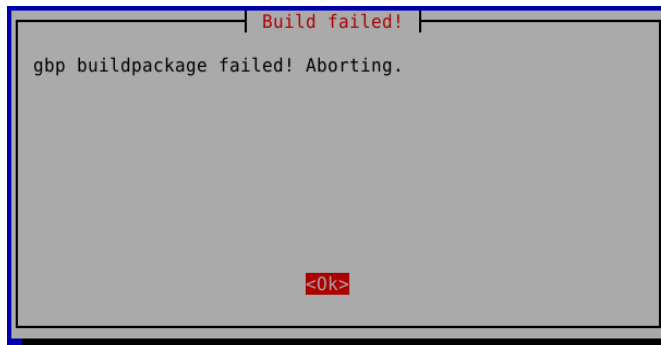


Abbildung 37.25.: Erfolgreicher Bauversuch!



## 38. Wenn das Bauen fehlschlägt

Wenn das Bauen fehlschlägt oder die nachfolgenden Tests (Kapitel 40, Seite 327) nicht zufriedenstellend ausfallen, kann das verschiedenartige Ursachen haben.

Erster Ansatzpunkt für die Ursachenforschung ist das Studium der Datei  
< *SourceName* >\_*Version* >-< *Revision* >\_*Arch* >.build.

Eine Ursache, warum das Bauen fehlgeschlagen ist, kann ein unzureichendes Ermitteln der Build-Abhängigkeiten sein. Die Ermittlung, ob benötigte Abhängigkeiten bereits paketiert sind, ist in Kapitel 10.3 (Seite 29) beschrieben.

Was zu tun ist, wenn das Fehlschlagen auf fehlenden oder fehlerhaften Patches beruht, wird in Kapitel 36.3 (Seite 274) erörtert.





## 39. Bauen jenseits von *Unstable (sid)*

### 39.1. Bauen für bereits offiziell freigegebene Distributionen

Zum Bauen für *Backports*- und *Proposed-Updates*-Paketten (Kapitel 22, Seite 79) hat sich für uns die folgende Vorgehensweise bewährt. Die folgenden Ausführungen gelten für *Oldstable* und *Oldoldstable* entsprechend.

Zunächst ist von der Aufgabenauswahl aus (Kapitel 33.5, Seite 177) ein neuer Git-Branch anzulegen (Kapitel 44.1, Seite 375). Dieser geht in der Regel vom Git-Zweig *debian/sid* bzw. *master* oder *main* aus.

Der Name dieses neuen Zweiges sollte das Ziel-Repositoryum des Paketes angeben (z.B. *debian/bookworm-bpo*).

Gibt es diesen Git-Zweig schon, kann dieser genutzt werden. Hierzu kann man folgende Befehle verwenden.

```
325 <Merge2Stable 325>≡
 git branch -vv
 git checkout <OlderBranch>
 git merge debian/sid # or master or main
 # Solve merge conflicts esp. d/changelog
 nano debian/changelog
 git add debian/changelog
 git commit
 # This is the merge commit
```

Die Behebung des Merge-Konfliktes erfordert in der Regel zumindest die Bearbeitung der Datei *debian/changelog*. Es wird ein neuer Eintrag in *debian/changelog* erstellt. (Kapitel 37.1, Seite 291)

Für den Versionseintrag in *debian/changelog* ist zwingend die in Kapitel 22.6 (Seite 83) beschriebene Nomenklatur zu verwenden. Diese hängt davon ab, für welchen Zweig nun gebaut wird.

Diese lauten:

**stable-proposed-updates** Es soll die bereits im Stable-Zweig vorhandene Paketversion bei der nächsten Zwischenveröffentlichung ersetzt werden.

**stable-updates** Es soll die bereits im Stable-Zweig vorhandene Paketversion dringend ersetzt werden.

**stable-backports** Zusätzlich zur älteren Paketversion soll eine neuere Version verfügbar gemacht werden.

## 39.2. Proposed-Updates – Besonderheiten

Der Changelog **darf nicht** die Nummer des Fehlerberichtes enthalten, der gegen *release.debian.org* erstellt wurde. Er sollte aber die Nummer des Fehlerberichtes enthalten, der gegen dieses **Debian**-Paket erstellt wurde und weswegen diese Version gebaut wird.

Danach wird das Paket für *proposed-updates* gebaut. Es ist eine neue Revision zu bauen (Kapitel 35, Seite 225). Hierzu müssen gegebenenfalls in der Datei *debian/control* die Versionen für die Abhängigkeiten angepasst werden.

Nach dem Bauen für die Veröffentlichung (Kapitel 41, Seite 343) und **vor** dem tatsächlichen Hochladen mit *dput* (Kapitel 43, Seite 357) wird mit *debdiff* (Kapitel 40.6.1, Seite 335) noch die Differenz zwischen der bisherigen Version im angestrebten Zweig und der neuen Version erstellt (Kapitel 40.6.1, Seite 335).

Wenn das Release-Team dem Antrag zustimmt, kann das Paket gebaut und hochgeladen werden. (Kapitel 42.1, Seite 351 bzw. Kapitel 43.3, Seite 362). Das Paket landet dann in der entsprechenden *New-Queue*.

Der Fehlerbericht an *release.debian.org* wird geschlossen, sobald das Paket dem nächsten Pointrelease der stabilen Version hinzugefügt wurde. Dies ist dann das Ende dieses Prozesses.

## 40. Überprüfungen

Eine erste Qualitätskontrolle erfolgt bereits während des Bauvorganges im *pbuilder* mit *lintian*.

Die nachfolgende Qualitätskontrolle erfolgt erneut mit *lintian* im *pedantic*-Modus (Kapitel 40.3, Seite 330).

```
327a <RunningTests 327a>≡ (334b)
function RunningTests {
 # Called by TaskSelect BuildNewRevision

 # QA using lintian and uscan

 # lintian
 RunningLintian
<RunningTests1 327b>
```

und mit *uscan* hinsichtlich des Inhaltes der Datei *debian/watch* (Kapitel 40.4, Seite 332).

```
327b <RunningTests1 327b>≡ (327a)
uscan
if [$!linc -eq 0]
then
 RunningUscan
else
 usq=1
fi
```

<RunningTests3 327c>

Wird mindestens eines der beiden Testergebnisse als mangelhaft qualifiziert, beendet sich das Programmskript nach einem entsprechenden Eintrag in die Log-Datei.

Andernfalls erfolgt die Frage, ob das Hochladen des Paketes vorbereitet werden soll.

```
327c <RunningTests3 327c>≡ (327b)
if [$usq -ne 0] || [$!linc -ne 0]
then
 echo "At least one test failed!" >> ${log}
 exit
else
 if whiptail --title "Upload?" \
 --yesno "Should the package be prepared to be uploaded now?" \
 --yes-button "Yes" --no-button "Exit" 15 60
 then
<RunningTests4 328>
```

4. Mai 2025

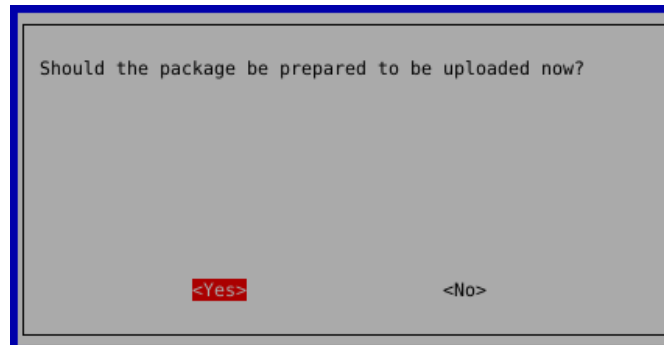


Abbildung 40.1.: Upload des Release vorbereiten

Wird die Frage bejaht, wird das Hochladen vorbereitet (Kapitel 41, Seite 343).

```
328 <RunningTests4 328>≡ (327c)
 Task=6 # Go to PrepareUploading
 else
 exit
 fi
fi
}
<CheckRepackSuffix 223a>
```

Später folgen noch weitere Überprüfungen mit *debdiff* (Kapitel 40.6.1, Seite 335).und *diffoscope*. Schließlich wird auch noch *piuparts* beschrieben.

## 40.1. Auswahl der Changes-Datei

Diese Funktion dient der Auswahl der Changes-Datei (*\*.changes*), die zur Prüfung des Paket-Bauergebnisses und zur Bestimmung der hochzuladenen Dateien genutzt wird.

```
329 <SelectChangesFile 329>≡ (247a)
function SelectChangesFile {
 # Called by RunningLintian SelectUploadTarget

 titlestr=${1}
 cd ${PrjPath}
 changesa=(ls ${SourceName}_${Version1}*_*.changes))

 if [-z "${changesa}"]
 then
 echo "File *"${SourceName}"*_ "${Version1}"*_*.changes not found!"
 exit
 fi

 i=0; slct=''
 for element in ${changesa[*]}
 do
 slct=${slct}' $i' '${element}' off '
 i=$(expr $i + 1)
 done

 paket=$(whiptail --title "${titlestr}" --radiolist "Select:" \
 --cancel-button "Exit" 15 60 8 $slct 3>&2 2>&1 1>&3)

<SelectChangesFile1 330a>
```

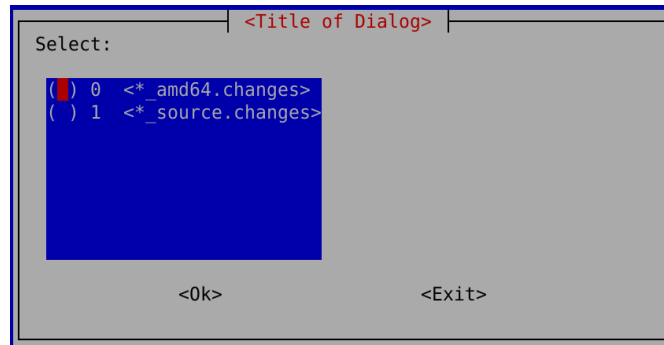


Abbildung 40.2.: \*.changes-Datei auswählen

Erfolgt keine Auswahl beendet sich das Programmskript.

```
330a <SelectChangesFile1 330a>≡ (329)
 if [-z "${paket}"]
 then
 exit
 fi
}
<PatchHeader 288>
```

## 40.2. Yamllint

Mit dem Kommandozeilentool *yamllint* kann die Syntax der *.yaml*-Dateien geprüft werden.

## 40.3. Prüfung mit Lintian

Das Ergebnis des Bauens wird mit *lintian* überprüft. Hierzu ist zunächst die *\*.changes*-Datei auszuwählen, auf die sich die Prüfung beziehen soll. Es wird daher die Funktion *SelectChangesFile* aufgerufen (Kapitel 40.1, Seite 329).

```
330b <RunningLintian 330b>≡ (321)
 function RunningLintian {
 # Called by RunningTests

 SelectChangesFile "lintian_check" # String will be found in ${1}
 linfile=${changesa[${paket]}}
 }
 <RunningLintian1 330c>
 Lintian wird mit Optionen aufgerufen, die eine ausführliche Prüfung bewirken.
330c <RunningLintian1 330c>≡ (330b)

 set +e
 lininfo=$(lintian --check --display-experimental --display-info \
--info --verbose --show-overrides --pedantic --tag-display-limit 0 \
--color auto ${linfile})
 lx=${?}

 <RunningLintian3 331a>
```

Wenn *Lintian* nichts meldet, soll der Nutzer dieses erfreuliche Ergebnis erfahren.

```
331a <RunningLintian3 331a>≡ (330c)
 if [${lx} -eq 0]
 then
 lininfo="Lintian does not find any Errors \
 \n\n Congratulations \n\n"${lininfo}
 fi
```

<RunningLintian4 331b>

Die Variable *lininfo* muss noch mit *sed* bearbeitet werden, damit die *Lintian*-Meldungen in einzelnen Zeilen erscheinen.

```
331b <RunningLintian4 331b>≡ (331a)
 # Make lininfo better readable
 lininfo=$(echo ${lininfo} | sed --expression='s/ [EWIPNX]:/\n&/g')

 set -e
 echo -e "lintian("${lx}"): "${lininfo}
 whiptail --title "Lintian" --msgbox "${lininfo}" --scrolltext 15 60
 echo -e "Result of Lintian:\n"${lininfo} >> ${log}
```

<RunningLintian5 331c>

Nach der Anzeige des Ergebnisses der Prüfung ist das Ergebnis vom Nutzer zu bewerten.

```
331c <RunningLintian5 331c>≡ (331b)
 whiptail --title "All well?" \
 --yesno "All well? Continue?" --yes-button "Yes" \
 --no-button "Exit" 15 60
 liq=$?
 }
```

<RunningUscan 332>

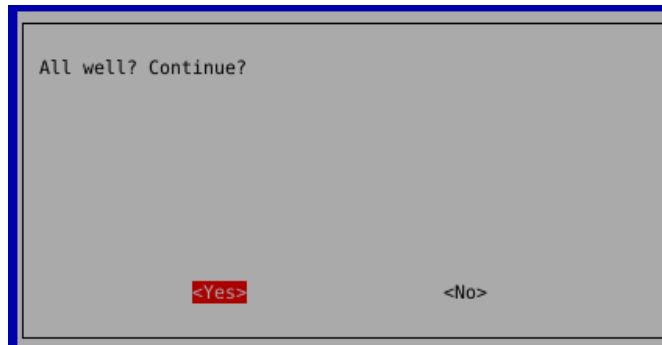


Abbildung 40.3.: Lintian: All Well?

Ist das Ergebnis in Ordnung, folgt die nächste Prüfung. Hinweise zur Fehlerbehebung sind im Kapitel 28 (Seite 97) zu finden.

## 40.4. Uscan

Mit der Option `-verbose` erstellt `uscan` einen für Menschen lesbaren Bericht über die Programmausführung. Mit der Option `-debug` wird zusätzlich der Status der internen Variablen angezeigt.

```

332 <RunningUscan 332>≡ (331c)
 function RunningUscan {
 # Called by RunningTests BuildWithUscan

 cd ${GitPath}

 # Check whether debian/watch exists
 if [! -e debian/watch]
 then
 whiptail --title "No debian/watch file found" \
 --yesno "Is there a good reason for having no debian/watch file?" \
 --defaultno --yes-button "Yes" --no-button "No" 15 60
 if [$? -eq 0]
 then
 usq=0
 else
 usq=1
 fi
 return
 fi

 set +e
 uscaninfo=$(uscan --no-download --verbose)
 if [${#uscaninfo} -gt 0]
 then
 whiptail --title "uscan" --msgbox "${uscaninfo}" --scrolltext 15 60
 echo -e "Result of uscan:\n"${uscaninfo} >> ${log}
 <RunningUscan1 333a>

```



Hier wird aus *uscaninfo* ausgelesen, ob die gebaute Version auch die aktuelle ist. Dies gilt für alle Builds, die in *experimental* oder *sid* veröffentlicht werden sollen. Die dazugehörige Meldung lautet: -> Package is up to date ...".

```
333a <RunningUscan1 333a>≡ (332)
 echo ${uscaninfo} | grep '=> Package is up to date' > /dev/null
 usc1=$?
 echo ${uscaninfo} | grep '=> Only older package available' > /dev/null
 usc2=$?
 if [${usc1} -eq 0]
 then
 whiptail --title "uscan" --msgbox "Package seems to be up to date." 15 60
 usq=0
<RunningUscan2 333b>
```

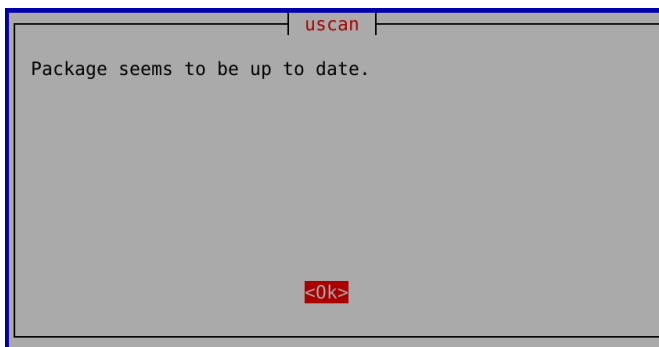


Abbildung 40.4.: Paket ist aktuell

```
333b <RunningUscan2 333b>≡ (333a)
 elif [${usc2} -eq 0]
 then
 whiptail --title "uscan" --msgbox "Only older package available." 15 60
 usq=0
<RunningUscan3 334a>
```

4. Mai 2025

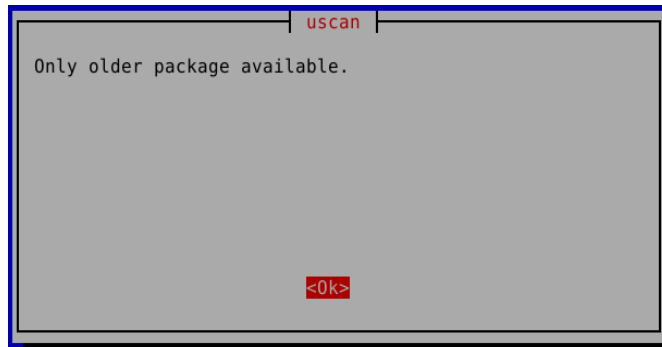


Abbildung 40.5.: Älteres Paket verfügbar

```
334a <RunningUscan3 334a>≡ (333b)
 else
 whiptail --title "No up to date message" \
 --yesno "No up to date message.\nRegardless all well? Continue?" \
 --defaultno --yes-button "Yes" --no-button "Exit" 15 60
 usq=$?
 fi
<RunningUscan4 334b>
```

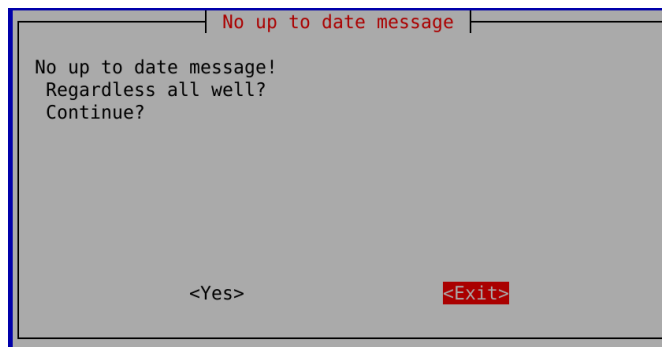


Abbildung 40.6.: Uscan - OK?

```
334b <RunningUscan4 334b>≡ (334a)
 else
 echo "uscan failed" >> ${log}
 whiptail --title "uscan failed" --msgbox "uscan failed" 15 60
 usq=1
 fi
 set -e
 }

<RunningTests 327a>
```

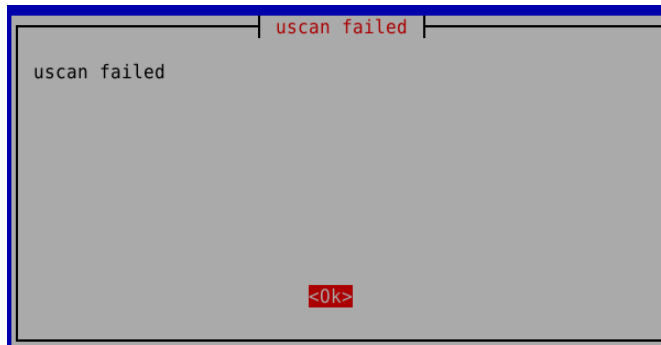


Abbildung 40.7.: Uscan schlägt fehl

## 40.5. Überprüfen der Datei *debian/copyright*

An dieser Stelle wird überprüft, ob auch wirklich **alle** Angaben zu den Lizenzen in der Datei *debian/copyright* erfolgt sind. Dazu gibt es mehrere Werkzeuge. Diese werden in Kapitel 10.1 (Seite 27) beschrieben.

Derzeit muss diese Prüfung noch manuell erfolgen, denn der Gebrauch dieser Werkzeuge ist manchmal nicht ausreichend.

## 40.6. Überprüfen mit *debdiff* und *diffoscope*

Beide Programme dienen dazu, die Differenz zwischen dem aktuellen Paket und der Vorversion darzustellen.

*debdiff* wird mit dem Paket *devscripts* installiert. *diffoscope* dagegen muss mit dem gleichnamigen Paket zusätzlich installiert werden.

### 40.6.1. *debdiff*

Mit *debdiff* können zwei Debian-Quellcodepaketen verglichen werden.

*debdiff* dient in diesem Falle dazu zu belegen, dass zwischen zwei Quellpaketen keine oder nur geringe Differenzen vorhanden sind.

```

335 <DebDiff 335>≡ (336b)
 function DebDiff {
 # Called by TaskSelect Ask4DebDiff
 debdiffFlag=$1

 if [${debdiffFlag} -gt 0]
 then
 whiptail --title "debdiff" \
 --msgbox "Now you can detect the differences between two Debian packages.\n" \
 15 60
 fi
 }

```

<DebDiff1 336a>

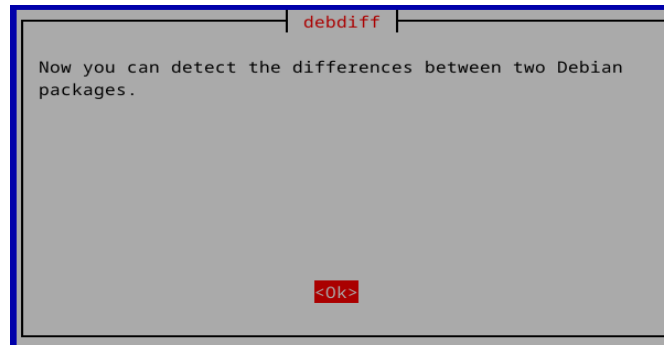


Abbildung 40.8.: Unterschiede feststellen

Der Nutzer kann zwei von ihm ausgewählte Quellcodepakete miteinander vergleichen, Hierzu werden *debdiff* die beiden Pakete in Form von *.dsc*-Dateien übergeben. Anhand dieser Dateien wird der Inhalt der Quellcodepakete verglichen.

```
336a <DebDiff1 336a>≡ (335)
 cd ${PrjPath}
 set +e
 PackageList=$(ls ${PrjPath} | grep \.dsc$ | sort --reverse --version-sort)
 set -e
 PackageArray=(${PackageList})

 i=0
 for element in ${PackageArray[*]}
 do
 packageE=${packageE}' '$i' '${element}' off '
 i=$((expr $i + 1))
 done

 <DebDiff4 337a>
```

Für diese Auswahlmöglichkeit werden alle lokal verfügbaren *.dsc-Dateien* geordnet angezeigt. Davon können dann genau die zwei Dateien zum Vergleich ausgewählt werden.

```
336b <DebDiffCheckList 336b>≡ (133a)
 function DebDiffCheckList {
 # Called by DebDiff

 PackageNrL=$(whiptail --title "debdiff" \
 --checklist "Which two versions should be compared?" \
 15 60 8 \
 ${packageE} --cancel-button "Cancel" 3>&2 2>&1 1>&3)

 PackageNrA=(${PackageNrL})
 }

 <DebDiff 335>
```

Das Programmskript prüft nun, ob genau zwei Einträge (*.dsc*-Dateien) ausgewählt wurden.

```
337a <DebDiff4 337a>≡ (336a)
 DebDiffCheckList
```

```
 if [${#PackageNrA[@]} -gt 2]
 then
 whiptail --title "Too much selections" \
 --msgbox "Please select only two versions" 15 60
 DebDiffCheckList
```

```
<DebDiff5 337b>
```

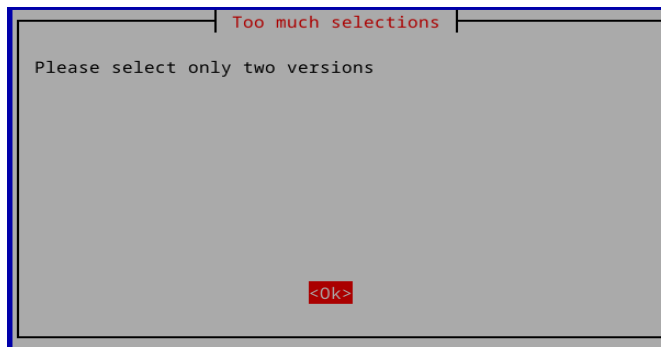


Abbildung 40.9.: Zu viele Versionen ausgewählt

```
337b <DebDiff5 337b>≡ (337a)
 elif [${#PackageNrA[@]} -lt 2]
 then
 whiptail --title "Less selections" \
 --msgbox "Please select two versions" 15 60
 DebDiffCheckList
 fi
```

```
<DebDiff6 338>
```

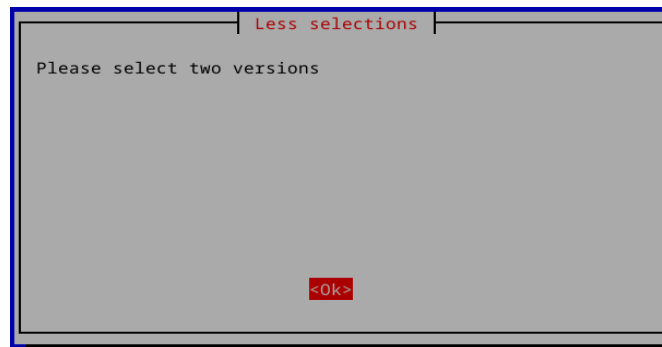


Abbildung 40.10.: Zu wenig ausgewählt

Wurden genau zwei *.dsc*-Dateien ausgewählt, erfolgt der Vergleich der Quellcodepakete mit *debdiff*.

338  $\langle \text{DebDiff6 } 338 \rangle \equiv$  (337b)

```

if [-z "${PackageNrL}"]
then
 cd ${GitPath}
 CommonTasks
fi

if whiptail --title "Reverse order?" \
 --yesno "Should the order of the packages be reversed?" \
 --yes-button "Yes" --no-button "No" --defaultno 15 60
then
 sn=${PackageNrA[1]}
 fn=${PackageNrA[0]}
else
 fn=${PackageNrA[1]}
 sn=${PackageNrA[0]}
fi

```

$\langle \text{DebDiff7 } 339a \rangle$

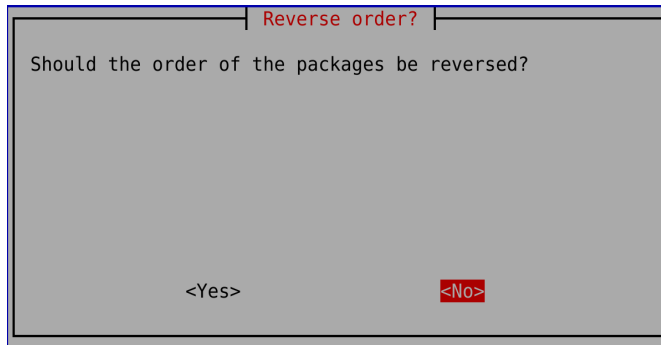


Abbildung 40.11.: Umkehr der Reihenfolge der zu vergleichenden Pakete

In bestimmten Fällen ist es sinnvoll, die Vergleichsreihenfolge umzukehren.

Regelmäßig ist das Resultat dann richtig, wenn im Dateinamen die ältere Version vor der neueren erscheint.

339a `<DebDiff7 339a>≡` (338)

```
Killing toxic quotationmarks
fn=$(echo $fn | sed --expression="s/\\/\\/g")
sn=$(echo $sn | sed --expression="s/\\/\\/g")
```

`<DebDiff8 339b>`

Die Ausgabe von `debdiff` erfolgt in eine entsprechend benannten Datei im Projektverzeichnis. Der Inhalt dieser Datei wird mit `less` angezeigt.

339b `<DebDiff8 339b>≡` (339a)

```
echo -e "Compare with debdiff: \n" \
 ${PackageArray[fn]} and "${PackageArray[sn]}"\n" >> \
 ${log}
echo -e "The result can be found in\n" \
 debdiff_${PackageArray[fn]}-${PackageArray[sn]}.diff >> ${log}
set +e
debdiff --diffstat ${PackageArray[fn]} ${PackageArray[sn]} > \
 debdiff_${PackageArray[fn]}-${PackageArray[sn]}.diff
set -e
less debdiff_${PackageArray[fn]}-${PackageArray[sn]}.diff

if [${debdiffFlag} -eq 0]
then
 cd ${GitPath}
 CommonTasks
fi
cd ${GitPath}
}
```

`<ImportDebianPackage 156c>`





**Teil IV.**

**Veröffentlichen**



# 41. Vorbereitungen zum Hochladen

Bisher haben wir uns damit beschäftigt, auf unserem Rechner ein Debian-Paket zu bauen, wie es auch vom Projekt Debian bereitgestellt wird.

Nun geht es darum, dass dieses Paket auch hochgeladen und damit dem Debian-Projekt zur Verfügung gestellt werden kann.

Die Vorbereitung erfolgt durch die Funktion *PrepareUploading*.

## 41.1. Existiert *debian/changelog*?

Es wird sicherheitshalber überprüft, ob eine Datei *debian/changelog* bereits existiert.

```
343a <PrepareUploading 343a>≡
 function PrepareUploading {
 # Called by TaskSelect

 cd ${GitPath}

 # Check debian/changelog
 if [-f debian/changelog]
 then
 less --LINE-NUMBERS debian/changelog
 else
 <PrepareUploading1 343b>

 Ist diese Datei nicht vorhanden, bricht das Programmskript an dieser Stelle mit
 einem entsprechenden Hinweis ab.

343b <PrepareUploading1 343b>≡ (343a)
 whiptail --title "This is the end" \
 --msgbox "No changelog - no upload!" 15 60
 exit
 fi

 <PrepareUploading2 344>
```

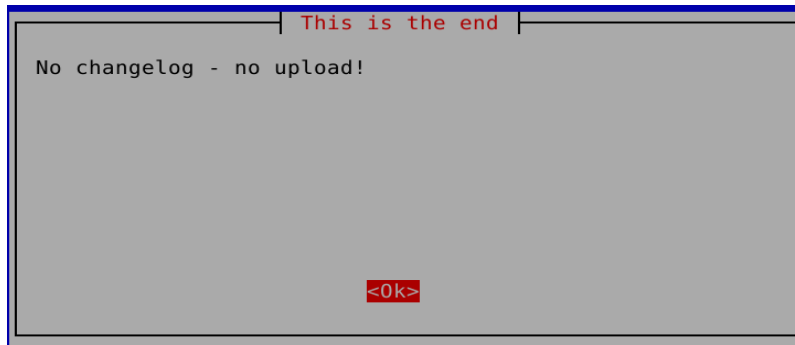


Abbildung 41.1.: Kein Changelog - kein Hochladen

Andernfalls wird der Changelog für eine Veröffentlichung vorbereitet oder verwandt. Hierzu wird abgefragt, ob der Changelog schon veröffentlichungsreif ist.

Dazu muss auf jeden Fall das „UNRELEASED“ durch die jeweilige Distributionsbezeichnung – meist „unstable“ – ersetzt werden.

```
344 <PrepareUploading2 344>≡ (343b)
 if ! whiptail --title "Changelog fit for publishing?" --defaultno \
 --yesno "Is the changelog fit for publishing?" --yes-button "Yes" \
 --no-button "No" 15 60
 <PrepareUploading3 345>
```

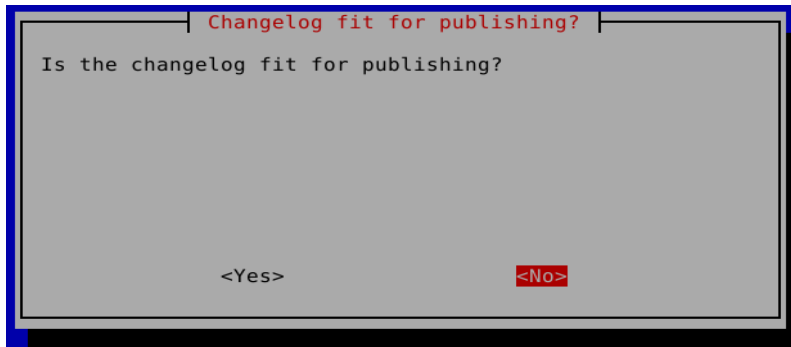


Abbildung 41.2.: Changelog veröffentlichungsreif?

Wird die Frage bejaht, wird die Option eröffnet, das Paket nochmal zu bauen (Kapitel 41.3, Seite 348).

## 41.2. *debian/changelog* fertigstellen

Ist die Datei *debian/changelog* nicht veröffentlichungsreif, wird sie verbessert.

Hierzu wird zunächst dazu aufgefordert zu prüfen, ob man im richtigen **Git**-Zweig ist. Mit der Funktion *AskDist* (Kapitel 37.3.1, Seite 302) wird ermittelt, welches der aktuelle **Git**-Zweig und welche Distribution ihm zugeordnet ist.

Danach wird das Ergebnis angezeigt. Gegebenenfalls kann in einem weiteren Terminal der Zweig gewechselt werden.

```
345 <PrepareUploading3 345>≡ (344)
 then
 AskDist
 echo -e "Notice from PrepareUploading: The branch is "${RecentBranch}"\n \
The distribution is "${RecentBranchD}" >> ${log}
 whiptail --title "Please check! (U)" \
 --msgbox "The branch is ${RecentBranch}" 15 60
 <PrepareUploading4 346>
```

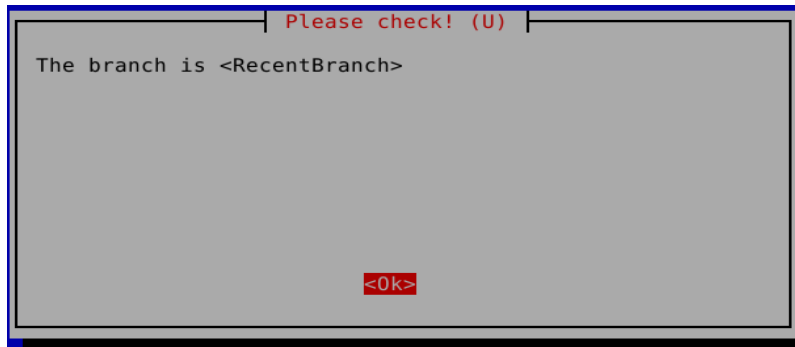


Abbildung 41.3.: Branch überprüfen

Sodann ist gegebenenfalls der Name der Distribution anzugeben, wohin das gebaute Debian-Paket hochgeladen werden soll. In der Regel ist dies die Distribution *unstable*.

346 `<PrepareUploading4 346>≡` (345)

```
if ["${RecentBranchD}" = "sid"]
then
 distName="unstable"
elif ["${RecentBranchD}" = "experimental"]
then
 distName="experimental"
else
 distName=$(whiptail --title "Name of the distribution" \
--inputbox "Please insert the name of the distribution\n \
specified in the changelog" \
--cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
fi
<PrepareUploading5 347a>
```

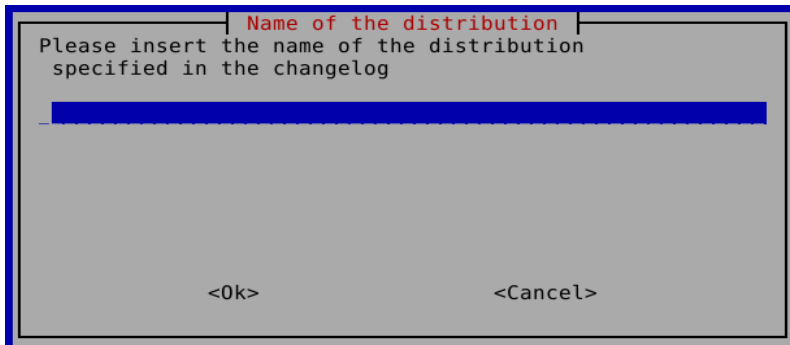


Abbildung 41.4.: Name der Distribution eingeben

Wird kein Name eingegeben, wird *unstable* als Distribution angenommen.

Dann wird nochmal der Name der Distribution mit der Aufforderung zur Prüfung angezeigt.

```
347a <PrepareUploading5 347a>≡ (346)
 if [-z "${distName}"]
 then
 distName="unstable"
 fi

 echo -e "Another notice from PrepareUploading:\n \
The distribution is now "${distName}" >> ${log}
 whiptail --title "Please check! (U)" \
 --msgbox "The distribution is ${distName}" 15 60
 <PrepareUploading6 347b>
```



Abbildung 41.5.: Name der Distribution überprüfen

Gerade bei neuen Paketen verwenden die Autoren gerne einen Upload nach *experimental*. Für neue Pakete ist nämlich ein *Binary*-Upload notwendig.

```
347b <PrepareUploading6 347b>≡ (347a)

 # making debian/changelog fit for publishing
 gbp dch --release --verbose --debian-branch=${RecentBranch} \
 --distribution=${distName} #--commit
 <PrepareUploading8 348a>
```

4. Mai 2025

Nach der Erstellung des Changelogs für das Release wird zur Kontrolle automatisch der Standardeditor geöffnet. Dies kann nicht vom Skript beeinflusst werden.

Diese Anzeige ist auch sinnvoll. Oft sind nämlich doppelte Commit-Einträge zu löschen.

```
348a <PrepareUploading8 348a>≡ (347b)
 git add .
 git commit -a

 whiptail --title "Build again" \
 --msgbox "Now the release will be built another time." 15 60
```

<PrepareUploading10 348b>

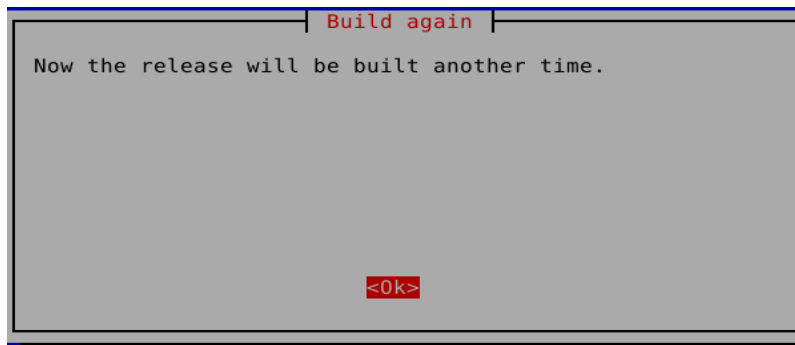


Abbildung 41.6.: Fürs Release bauen

Nun wird das Paket fürs Hochladen ins Debian-Repository gebaut.

Das nochmalige Bauen ist notwendig, da die Datei *debian/changelog* geändert wurde und daher erneut integriert werden muss.

```
348b <PrepareUploading10 348b>≡ (348a)
 # Building revision
 GpgKeyAvailable
 SBuildOrPBuilder 1
 if [$gbpq -eq 0]
 then
 echo "Package ${SourceName} was built using gbp buildpackage." >> ${log}
 fi
```

<PrepareUploading11 348c>

### 41.3. Nochmaliges Bauen?

Hier geht es weiter, wenn die Frage bejaht wird, ob die Datei *debian/changelog* veröffentlichungsreif ist. Dann wird die Option eröffnet, das Paket nochmals zu bauen.

Das Bauen erfolgt dann in der soeben beschriebenen Weise.

```
348c <PrepareUploading11 348c>≡ (348b)
 else
 if whiptail --title "Building another time?" \
 --yesno "Should the release be built another time?\n(Without tagging)" \
 --yes-button "Yes" --no-button "No" 15 60
```

<PrepareUploading12 349a>



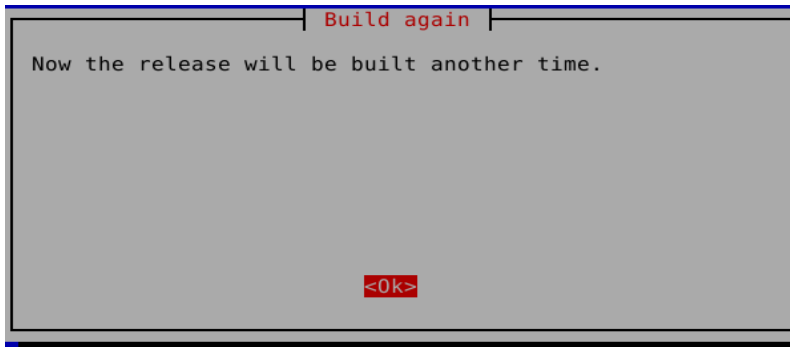


Abbildung 41.7.: Fürs Release bauen

```

349a <PrepareUploading12 349a>≡ (348c)
 then
 # Building revision
 SBuildOrPBuilder 2
 if [$gbpq -eq 0]
 then
 echo "Package ${SourceName} was built using gbp buildpackage." >> ${log}
 fi
 fi
 fi
 }

 <SelectUploadTarget 358a>

```

## 41.4. Soll ein Debdiff erstellt werden?

Bevor nach *salsa.debian.org* hochgeladen wird, wird gegebenenfalls abgefragt, ob ein *debdiff* (Kapitel 40.6.1, Seite 335). erstellt werden soll.

```

349b <TaskSelect9 349b>≡
 if [${rcts} -eq 0]
 then
 Ask4DebDiff
 <TaskSelect9-1 351a>

```

4. Mai 2025

Die Abfrage erfolgt nur, wenn jenseits vom Zweig *Unstable* paketiert werden soll (Kapitel 22, Seite 79).

```
350 <Ask4DebDiff 350>≡
 function Ask4DebDiff {
 # Called by TaskSelect

 # Check whether branch is sid
 if [${RecentBranch} == "debian/sid"] || [${RecentBranch} == "master"] || \
 [${RecentBranch} == "main"]
 then
 return
 fi

 # Creating a Debdiff?
 if whiptail --title "DebDiff needed?" \
 --yesno "Do you want to create a debdiff?" \
 --yes-button "Yes" --no-button "No" 15 60
 then
 DebDiff 1
 fi
 }

<TaskSelect (nicht definiert)>
```

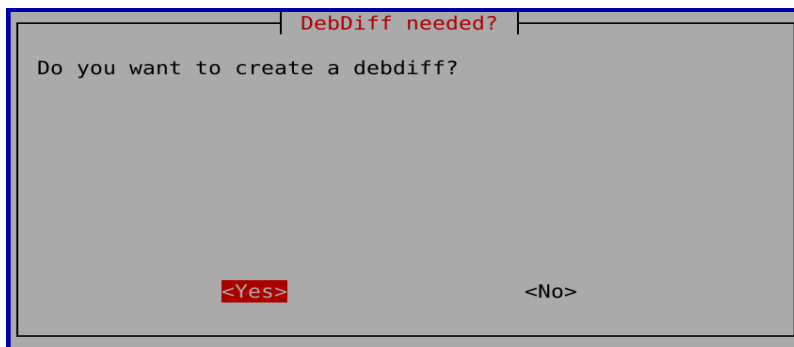


Abbildung 41.8.: DebDiff benötigt?

## 42. Hochladen auf Git-Repositories

Im folgenden Abschnitt aus der Aufgabenauswahl werden zunächst die Funktionen aufgerufen, um in die Git-Repositories hochzuladen.

```
351a <TaskSelect9-1 351a>≡ (349b)
 #####
 # Pushing git repo
 #####
 Upload2OwnServer
 Upload2Salsa
 <TaskSelect10 357a>
```

### 42.1. Hochladen nach salsa.debian.org

Vor dem Hochladen nach *salsa.debian.org* überprüft das Programmskript, ob dort überhaupt schon ein entsprechendes Repository vorhanden ist.

```
351b <Upload2Salsa 351b>≡ (355)
 function Upload2Salsa {
 # Called by TaskSelect and itself
 # Uploading to Salsa
 if whiptail --title "Upload to salsa.debian.org?" \
 --yesno "Should ${SourceName} be uploaded to Salsa?" \
 --yes-button "Yes" --no-button "No" 15 60
 then
 <Upload2Salsa1 352a>
```



Abbildung 42.1.: Fürs Release bauen

352a `<Upload2Salsa1 352a>≡` (351b)

```
BrowserName=$(echo ${SalsaName} | sed --expression='s/.git$//g')
BrowserName="https://salsa.debian.org/${BrowserName}
wget --spider --verbose --max-redirect=0 \
--append-output=${log} ${BrowserName}
if [$? -ne 0]
then
 whiptail --title "No project found at salsa.debian.org" \
--msgbox "Please create ${BrowserName} first" 15 60
 echo "No project "${BrowserName}" found at salsa.debian.org" \
>> ${log}

 if whiptail --title "Done?" \
--yesno "Created ${BrowserName} on salsa.debian.org?" \
--yes-button "Yes" --no-button "No" 15 60
 then
 Upload2Salsa
 else
 exit
 fi
else
```

`<Upload2Salsa5 352b>`

Wenn Patch-Queue-Zweige existieren, können diese vor dem Hochladen nach *salsa.debian.org* gelöscht werden.

352b `<Upload2Salsa5 352b>≡` (352a)

```
set +e
if echo $(git branch) | grep --quiet 'patch-queue/'
then
 if whiptail --title "Patch queue branches found:" \
--yesno "$(git branch | grep 'patch-queue')\n \
Delete all patch-queue branches?" \
--yes-button "Yes" --no-button "No" 15 60
```

`<Upload2Salsa6 353>`

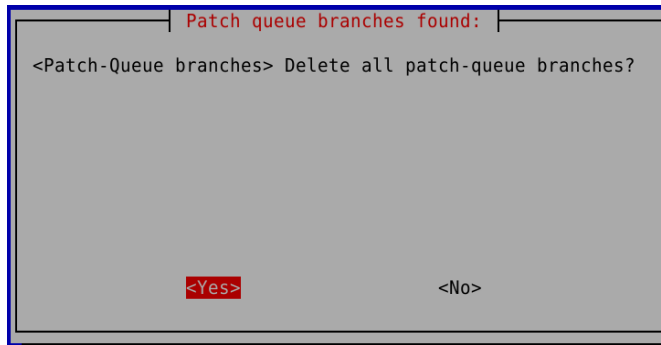


Abbildung 42.2.: Patch-Queue Zweige gefunden

```

353 <Upload2Salsa6 353>≡ (352b)
 then
 git branch --delete --force \
 $(git branch | grep 'patch-queue')
 fi
 fi
 set -e

 if ! whiptail --title "Last stop before upload!" \
 --yesno "Anything all right?" \
 --yes-button "Yes" --no-button "No" 15 60
 <Upload2Salsa7 354>

```

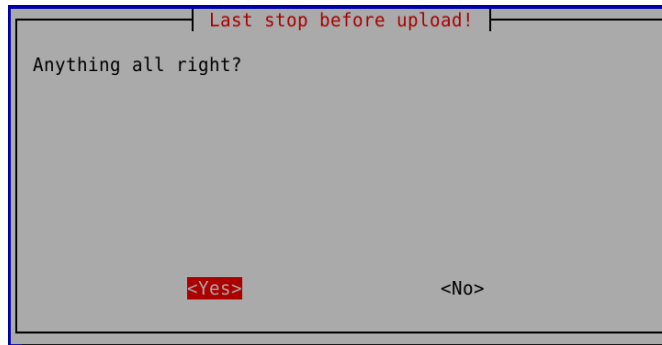


Abbildung 42.3.: Letzter Halt - Alles OK?

```

354 <Upload2Salsa7 354>≡
 then
 exit
 fi
 set +e
 if git remote | grep 'salsa' > /dev/null
 then
 RepoName="salsa"
 else
 RepoA=$(git remote)

 i=0; slct=''
 for element in ${RepoA[*]}
 do
 slct=${slct}' '$i' '${element}' off '
 i=$(expr $i + 1)
 done

 RepoNr=$(whiptail --title "Select repository" \
 --radiolist "Select one of these repositories" \
 --cancel-button "Exit" 15 60 8 \
 $slct 3>&2 2>&1 1>&3)

 if [-z "${RepoNr}"]
 then
 exit
 else
 RepoName=${RepoA[${RepoNr}]}
 fi
 fi

 git push --set-upstream ${RepoName} --all >> ${log}
 git push --set-upstream ${RepoName} --tags >> ${log}
 set -e

 echo "${SourceName}" was uploaded to salsa.debian.org." >> ${log}
 fi
 fi
}

<GettingFingerprint (nicht definiert)>

```

(353)

## 42.2. Hochladen auf eigenen Git-Server

Das Hochladen auf einen eigenen Git-Server setzt voraus, dass ein solcher eingerichtet wurde (Kapitel 20.4.2, Seite 74). Ferner ist zuvor sein Name oder seine IP-Adresse einzugeben (Kapitel 44.2, Seite 376).

```

355 <Upload2OwnServer 355>≡ (360b)
 function Upload2OwnServer {
 # Called by TaskSelect

 # Uploading to own git server
 if [-n "$ServerName"]
 then
 if whiptail --title "Upload to own git server?" \
 --yesno "Should ${SourceName} be uploaded to your own git server?" \
 --yes-button "Yes" --no-button "No" 15 60
 then
 git push --set-upstream home --all >> ${log}
 git push --set-upstream home --tags >> ${log}

 echo "${SourceName}" was uploaded to your git server." >> ${log}
 fi
 fi
 }

 <Upload2Salsa 351b>

```





## 43. Paket(e) hochladen

In der Funktion *TaskSelect* (Aufgabenauswahl) werden auch die Funktionen zum Hochladen der Pakete aufgerufen.

```
357a <TaskSelect10 357a>≡ (351a)
 #####
 # Uploading packages
 #####
 SelectUploadTarget
 CreateSignature
 UploadUsingDput
 Upload2PeopleD0
 UploadLocal
 <TaskSelect11 357b>
```

Wenn die Funktionen *CreateNewBranch* (Kapitel 44.1, Seite 375), *SelectBranch* (Kapitel 33.4, Seite 169) oder *OwnServer* (Kapitel 44.2, Seite 376) aufgerufen wurden, wird die Konfigurationsdatei erneut zur Bearbeitung angezeigt (Kapitel 33.1, Seite 165) und anschließend die Aufgabenauswahl erneut aufgerufen.

```
357b <TaskSelect11 357b>≡ (357a)
 else
 ConfigFileLEC
 CommonTasks
 fi
 }

 <StartTasks (nicht definiert)>
```

## 43.1. Auswahl des Zielrepositoriums

Vor dem Hochladen ist das Zielrepositorium auszuwählen. Dabei wird ein *Non-Maintainer-Upload* wie ein Repository behandelt.

```
358a <SelectUploadTarget 358a>≡ (349a)
function SelectUploadTarget {
 # Called by TaskSelect Upload2FtpMaster

 # Select upload target
 Upl=$(whiptail --title "Uploading?" \
 --radiolist "Should the package be uploaded to ftp-master,\n \
 people.d.o or mentors.debian.net?" 15 60 6\
 "0" "No" off \
 "1" "ftp-master" on \
 "2" "people.d.o" off \
 "3" "Mentors" off \
 "4" "Non-Maintainer-Upload" off \
 "5" "Local repository" off --cancel-button "Exit" 3>&2 2>&1 1>&3)
```

<SelectUploadTarget1 358b>

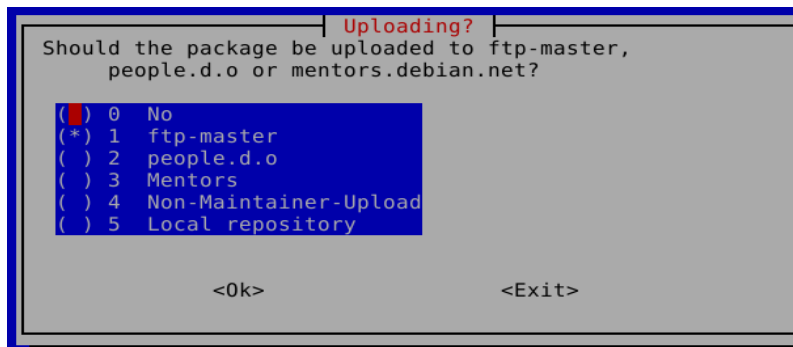


Abbildung 43.1.: Ziel des Uploads.

Wird „No“ oder „Cancel“ ausgewählt, wird das Programm beendet.

```
358b <SelectUploadTarget1 358b>≡ (358a)
The order of the conditions is important!
'Cancel' results an empty variable
if [-z ${Upl}] || [${Upl} -eq 0]
then
 exit
fi
```

<SelectUploadTarget2 359a>

Über den Wert, der der Variablen *Upl* zugewiesen wird, wird der weitere Ablauf gesteuert.

```
359a <SelectUploadTarget2 359a>≡ (358b)
 case "${Upl}" in
 1) Upltext="ftp-master";;
 2) Upltext="people.d.o";;
 3) Upltext="Mentors";;
 4) Upltext="delayed";;
 5) Upltext="local repository";;
 esac
```

<SelectUploadTarget3 359b>

Im Folgenden wird nun mit der Funktion *SelectChangesFile* (Kapitel 40.1, Seite 329) die *\*.changes*-Datei des hochzuladenen Paketes ausgewählt.

```
359b <SelectUploadTarget3 359b>≡ (359a)

 cd ${PrjPath}

 # Select package
 SelectChangesFile "Upload" # String will be found in ${1}
 UplPaket=${changesa[$paket]}

 # Version2=$(echo ${UplPaket} | sed --expression="s/^[a-z\-*_//]" | \
 # sed --expression="s/-.*///")
 # SourceName1=$(echo ${UplPaket} | sed --expression="s/_.*//1")
 # OrigPaket=${SourceName1}"_"${Version2}".orig"
```

<SelectUploadTarget4 359c>

Vor dem Hochladen erfolgt noch eine Kontrollfrage, ob das ausgewählte Paket zum ausgewählten Ziel hochgeladen werden soll. Dabei wird zum ersten Mal der Wert *Upltext* benötigt.

```
359c <SelectUploadTarget4 359c>≡ (359b)
 # Final question before uploading starts
 if [${Upl} -ne 4]
 then
 if ! whiptail --title "Upload to ${Upltext}?" \
 --yesno "Do you want to upload ${UplPaket} to ${Upltext}." \
 --yes-button "Yes" --no-button "Exit" 15 60
 <SelectUploadTarget5 360a>
```

4. Mai 2025

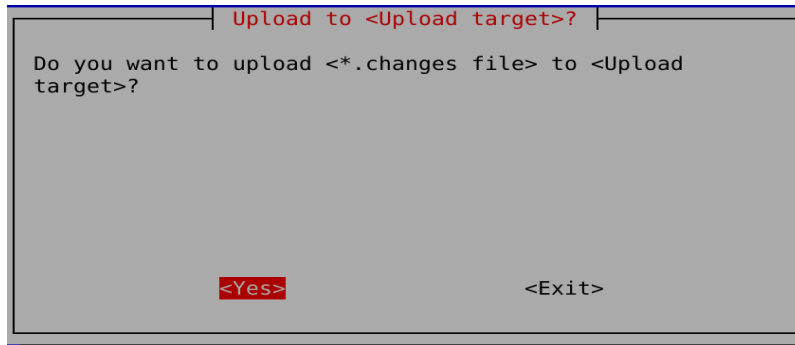


Abbildung 43.2.: Ist das angegebene Ziel des Uploads korrekt?

```
360a <SelectUploadTarget5 360a>≡ (359c)
 then
 whiptail --title "Bye" --msgbox "Bye" 15 60
 <SelectUploadTarget6 360b>
```

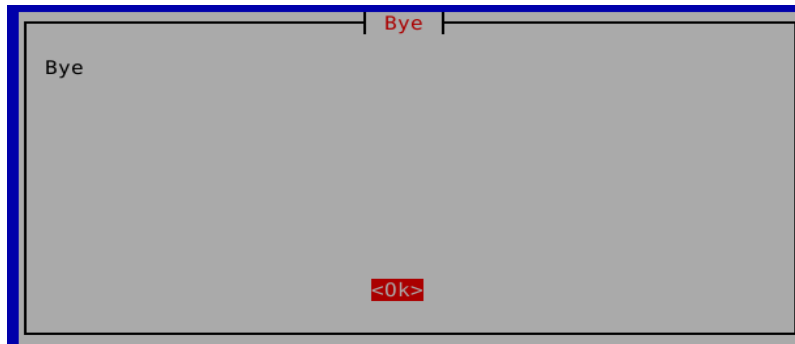


Abbildung 43.3.: Bye

```
360b <SelectUploadTarget6 360b>≡ (360a)
 exit
 fi
 fi

 echo "${UplPaket} should be uploaded to ${Upltext}." >> ${log}
}

<Upload2OwnServer 355>
```

## 43.2. Signatur erzeugen

Vor dem Hochladen muss das Paket signiert werden.

Die Funktion `CreateSignature` erzeugt die erforderlichen Signaturen mittels `debsign`. `debsign` signiert ein Debian-`.changes`- und `.dsc`-Dateipaar mittels GnuPG. Dazu muss der GnuPG-Schlüssel zur Verfügung stehen (Kapitel 32.8, Seite 160).

Wurde die Datei `*.changes` bereits signiert, erscheint auf der Konsole folgende Meldung:

```
The .changes file is already signed.
Would you like to use the current signature? [Yn]
```

Im Falle des Fehlschlagens wird eine Wiederholung angeboten.

```
361a <CreateSignature 361a>≡
function CreateSignature {
 # Called by TaskSelect Upload2FtpMaster and itself

 # Key available?
 GpgKeyAvailable

 # Signature using debsign
 set +e
 debsign -k${fipr} ${UplPaket}
 if [$? -ne 0]
 then
 if whiptail --title "Signing failed!" \
 --yesno "Signature failed - Retry?" \
 --yes-button "Yes" --no-button "Exit" 15 60
 <CreateSignatur3 361b>
```

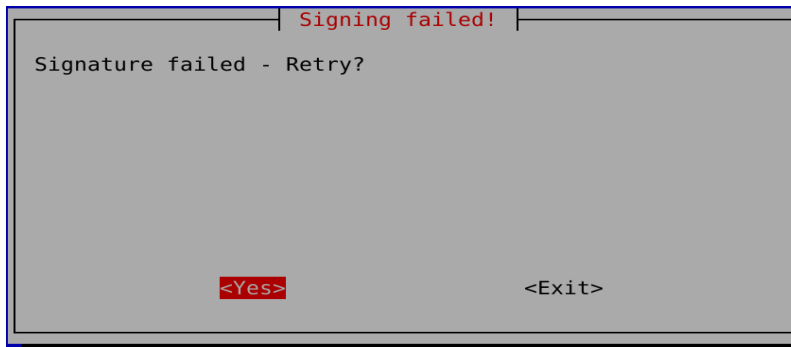


Abbildung 43.4.: Signieren erfolgreich?

```
361b <CreateSignatur3 361b>≡ (361a)
 then
 CreateSignature
 else
 exit
 fi
fi
 set -e
 echo "${UplPaket} was signed" >> ${log}
}

<Upload2Mentors 367>
```

### 43.3. Hochladen mit dput

In der Funktion *UploadUsingDput* erfolgt nur die Weichenstellung, ob das Paket mittels *dput* zu den Ftp-Mastern (Kapitel 43.4, Seite 362) oder nach *mentors.debian.net* (Kapitel 43.5, Seite 367) hochgeladen werden soll.

```
362a <UploadUsingDput 362a>≡ (365a)
function UploadUsingDput {
 # Called by TaskSelect Upload2FtpMaster

 # Uploading using dput

 cd ${PrjPath}/
 if [${Upl} -eq 3]
 then
 Upload2Mentors
 elif [${Upl} -eq 1] || [${Upl} -eq 4]
 then
 Upload2FtpMaster
 fi
}

<UploadFilesSelect 369b>
```

### 43.4. Nach FTP-Master hochladen

Die Pakete werden regelmäßig nach *ftp-master* hochgeladen, um sie durch das Debian-Projekt zu veröffentlichen. Andere Veröffentlichungswege sind eher die Ausnahme.

```
362b <Upload2FtpMaster 362b>≡ (366b)
function Upload2FtpMaster {
 # Called by UploadUsingDput

 # repeat question
 if whiptail --title "Last exit" \
 --yesno "Should the package be uploaded to ftp-master?" \
 --yes-button "Yes" --no-button "Exit" 15 60
 <Upload2FtpMaster1 363>
```

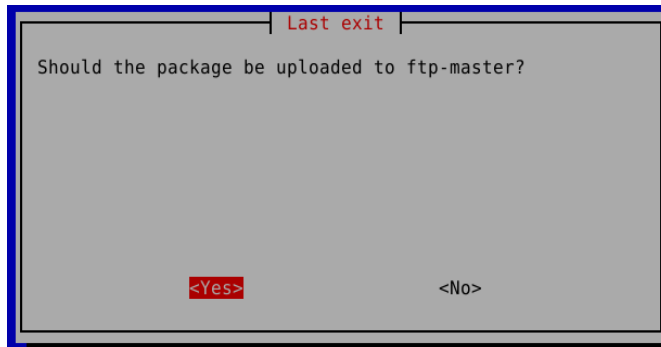


Abbildung 43.5.: Upload to FTP-Master - OK?

```

363 <Upload2FtpMaster1 363>≡ (362b)
 then
 set +e
 # Checking whether the .changes file is the right one for the upload target
 sourceFlag=$(echo ${UplPaket} | grep --count '_source.')
 expFlag=$(grep --line-number ') experimental; urgency=' ${GitPath}/debian/changelog \
 | grep '^1:')
 set -e
 echo -e "${UplPaket}:\n${expFlag}\nsourceFlag: ${sourceFlag}" >> ${log}
 # Strip line to isolate release
 expFlag=$(echo ${expFlag} | sed --expression='s/^.*) //' | \
 sed --expression='s/; .*$/')

 if [-z "${expFlag}"]
 then
 if [${sourceFlag} -eq 0]
 then
 if whiptail --title "Uploading?" \
 --yesno "Do you really want to upload a binary package\n \
 to ftp-master?" --yes-button "Yes" --no-button "No" 15 60

```

<Upload2FtpMaster2 364>

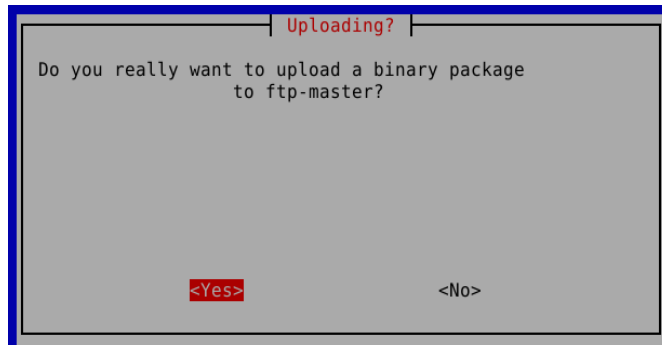


Abbildung 43.6.: Soll ein Binär-Paket auf FTP-Master hochgeladen werden?

```
364 <Upload2FtpMaster2 364>≡ (363)
 then
 Dput2FtpMaster
 else
 echo "Next try to upload" >> ${log}
 SelectUploadTarget
 fi
 else
 Dput2FtpMaster
 fi
 else
 if [$sourceFlag -ge 1]
 then
 if whiptail --title "Uploading?" \
 --yesno "Do you really want to upload a source package\n \
 to experimental?" --yes-button "Yes" --no-button "No" 15 60
 <Upload2FtpMaster3 365a>
```



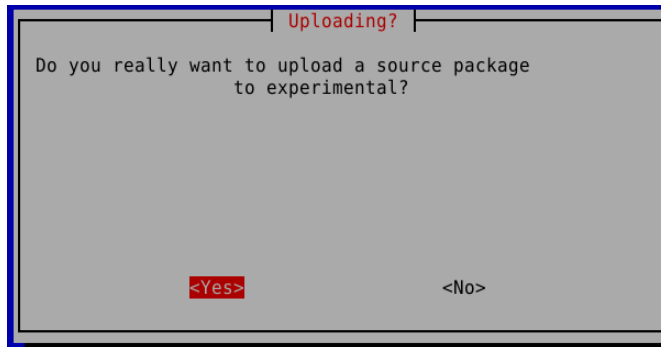


Abbildung 43.7.: Wirklich nach Experimental hochladen?

365a `<Upload2FtpMaster3 365a>≡` (364)

```

 then
 Dput2FtpMaster
 else
 echo "Next try to upload" >> ${log}
 SelectUploadTarget
 CreateSignature
 UploadUsingDput
 fi
else
 Dput2FtpMaster
fi
fi
}

```

`<UploadUsingDput 362a>`

In der folgenden Funktion erfolgt das Hochladen nach FTP-Master.

365b `<Dput2FtpMaster 365b>≡` (368b)

```

function Dput2FtpMaster {
 # Called by Upload2FtpMaster

 if whiptail --title "Simulate uploading?" \
 --yesno "Should the upload to ftp-master be simulated?" \
 --yes-button "Yes" --no-button "No" 15 60

```

`<Dput2FtpMaster1 366a>`

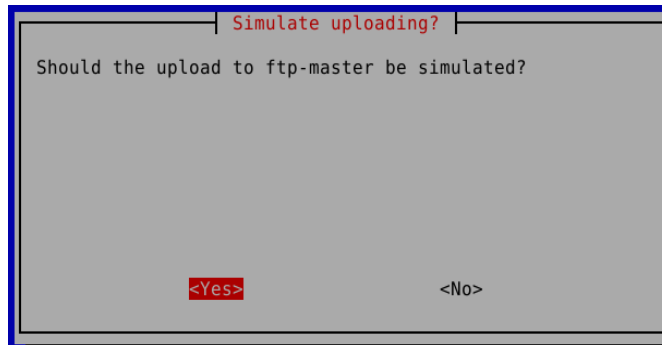


Abbildung 43.8.: Soll das Hochladen auf FTP-Master simuliert werden?

```
366a <Dput2FtpMaster1 366a>≡ (365b)
 then
 dput --simulate ftp-master ${UplPaket}
 echo
 echo "After reading press RETURN!"
 read x
 fi
```

```
if whiptail --title "Uploading to FTP-Master?" \
--yesno "Everything fine?\n\n \
Should the package be uploaded to ftp-master now?" \
--yes-button "Yes" --no-button "No" 15 60
```

```
<Dput2FtpMaster2 366b>
```

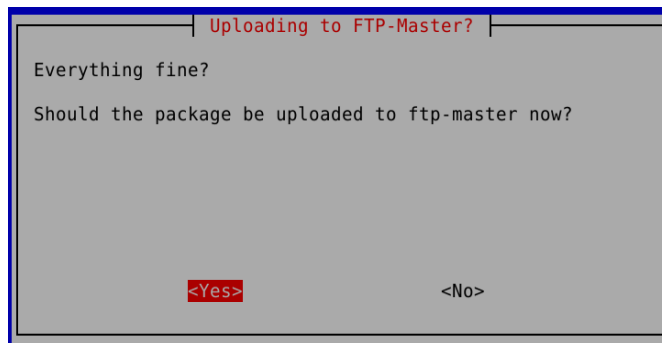


Abbildung 43.9.: Ist alles ok mit dem Hochladen?

```
366b <Dput2FtpMaster2 366b>≡ (366a)
 then
 if [${Upl} -eq 1]
 then
 dput ftp-master ${UplPaket}
 echo "${UplPaket} was uploaded to ${Upltext}." >> ${log}
 else
 Dput2NMU
 fi
 fi
}
```

```
<Upload2FtpMaster 362b>
```

Dieses Paket landet zunächst auf <https://incoming.debian.org/debian-buildd/pool/main/> bis es vom Build-Daemon (buildd) gebaut und zum Herunterladen zur Verfügung gestellt wird.

#### 43.4.1. Zurückweisung eines Paketes

Wenn ein Paket von den FTP-Mastern zurückgewiesen wird, sollte beim anschließenden Hochladen des korrigierten Paketes die Revisionsnummer nicht erhöht werden.

Dazu muss die Datei `<PaketName>_<Version>_source.ftp-master.upload` vorher im Projektverzeichnis entfernt werden.

### 43.5. Nach mentors.debian.net hochladen

```

367 <Upload2Mentors 367>≡ (361b)
function Upload2Mentors {
 # Called by UploadUsingDput

 if whiptail --title "Simulate uploading?" \
 --yesno "Should the upload to Mentors be simulated?" \
 --yes-button "Yes" --no-button "No" 15 60
 then
 dput --simulate mentors ${UplPaket}
 echo
 echo "After reading press RETURN!"
 read x
 fi

 # repeat question
 if whiptail --title "Uploading?" \
 --yesno "Should the package be uploaded to Mentors?" \
 --yes-button "Yes" --no-button "No" 15 60
 then
 dput mentors ${UplPaket}
 echo "${UplPaket} was uploade to ${Upltext}." >> ${log}
 fi
}

<Dput2NMU 368a>

```

## 43.6. Als *Non-Maintainer-Upload* hochladen

Im Rahmen der Ausbesserung veröffentlichungskritischer Fehler durch Andere als dem Paket-Maintainer ist es üblich, dem Paket-Maintainer noch eine gewisse Zeit zu lassen, selber das Problem zu lösen.

```
368a <Dput2NMU 368a>≡ (367)
function Dput2NMU {
 # Called by Dput2FtpMaster

 DelayDays=$(whiptail --title "Non-Maintainer-Upload" \
 --radiolist "Days for delay?" 15 60 5 \
 "0" " 5 days of delay" off \
 "1" "10 days of delay" on \
 "2" "15 days of delay" off --cancel-button "Exit" 3>&2 2>&1 1>&3)

 <Dput2NMU1 368b>
```

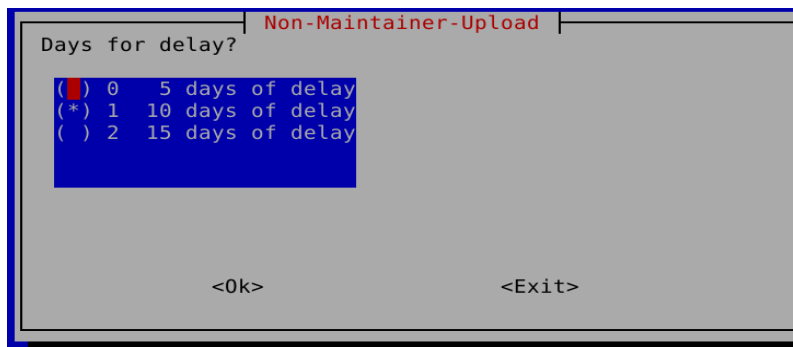


Abbildung 43.10.: Tage der Verzögerung

Es kann ausgewählt werden, wie viele Tage dem Paket-Maintainers zur eigenen Lösung des Problems verbleiben sollen.

```
368b <Dput2NMU1 368b>≡ (368a)
 if [-z ${DelayDays}]
 then
 exit
 fi

 case "${DelayDays}" in
 0) DelDays=5;;
 1) DelDays=10;;
 2) DelDays=15;;
 esac

 dput --delayed ${DelDays} ftp-master ${UplPaket}
}

<Dput2FtpMaster 365b>
```

## 43.7. Hochladen nach *people.debian.org*

```

369a <Upload2PeopleDO 369a>≡ (369b)
 function Upload2PeopleDO {
 # Called by TaskSelect

 if [${Upl} -eq 2]
 then
 # For people.d.o you can not use dput
 if whiptail --title "Archive on people.d.o" \
 --yesno "Does the directory public_html/${OrigName} \
 already exist at people.d.o?\n \
 If not you have to enter the following commands\n \
 in a separate terminal:\n\n \
 ssh <user>@people.debian.org\n \
 mkdir --parents public_html/${OrigName}" \
 --yes-button "Yes" --no-button "No" 15 60
 then
 UploadFilesSelect
 <Upload2PeopleDO3 370>
369b <UploadFilesSelect 369b>≡ (362a)
 function UploadFilesSelect {
 # Called by Upload2PeopleDO UploadLocal

 set +e
 UplFL=$(cat ${UplPaket} | grep --after-context=10 'Files: *')
 UplFL1=$(echo ${UplFL} | sed --expression='s/Files: //'')
 i=1
 while [$i -lt 6]
 do
 c=$(expr ${i} * 5)
 UplFL2=${UplFL2}" "${echo $UplFL1 | cut --delimiter=" " -f${c})
 i=$(expr ${i} + 1)
 done
 set -e
 }

 <Upload2PeopleDO 369a>

```

```

370 <Upload2PeopleDO3 370>≡ (369a)
 if whiptail --title "Upload file?" \
 --yesno "Should the following files\n${UplFL2}\n \
 have to be uploaded?" --yes-button "Yes" \
 --no-button "No" 15 60
 then
 if [-z "${pdoaccount}"]
 then
 pdoaccount=$(whiptail --title "Account at people.debian.org" \
 --inputbox "Please insert the name of your account on\n \
 people.debian.org" \
 --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
 if [-z "${pdoaccount}"]
 then
 echo "Please insert the name of your account on\n \
 people.debian.org"
 read pdoaccount
 fi
 changeflag=1
 fi

 if ! whiptail --title "Account name" \
 --yesno "The name of your account on people.debian.org:\n \
 ${pdoaccount}" --yes-button "Yes" --no-button "No" 15 60
 then
 pdoaccount=$(whiptail --title " Account name" \
 --inputbox "Name of your account on people.debian.org:" \
 --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
 if [-z "${pdoaccount}"]
 then
 echo "Please insert the name of your account on\n \
 people.debian.org"
 read pdoaccount
 fi
 changeflag=1
 fi

 if [$changeflag -eq 1]
 then
 echo 'pdoaccount=${pdoaccount} >> ${ConfigPath}${OrigName}'
 changeflag=0
 fi

 cd ${ProjectPath}/${OrigName}
 if scp -p ${UplFL2} \
 ${pdoaccount}@people.debian.org:/home/${pdoaccount}/public_html/${OrigName}
 then
 echo "${UplFL2} were uploaded to p.d.o." >> ${log}
 else
 echo "Something went wrong while uploading to p.d.o." >> ${log}
 echo "Tried to execute this command:" >> ${log}
 echo "scp -p "${UplFL2}" "${pdoaccount}"@people.debian.org:/home/"\
 ${pdoaccount}"/public_html/"${OrigName} >> ${log}
 fi
 pdoarchivetext="If the archive on people.d.o should be\n \
 used too, you have to enter the following commands:\n \

```

```

ssh <user>@people.debian.org\n \
cd public_html/${OrigName}\n \
apt-ftparchive packages . > Packages\n \
apt-ftparchive sources . > Sources\n \
cat Packages | gzip -c > Packages.gz\n \
cat Sources | gzip -c > Sources.gz\n \
apt-ftparchive release . > Release"

if whiptail --title "Archive on people.d.o" \
--yesno "${pdoarchivetext}\n\n \
Do you like to copy and paste these commands?" \
--yes-button "Yes" --no-button "No" 15 60
then
 echo -e $pdoarchivetext
 echo -e "\nPlease press any key to continue!"
 read x
fi
fi
fi
}

<UpdateLocalRepo 372>

```

## 43.8. Lokales Repositorium

Es kommt immer wieder vor, dass Pakete gebaut werden müssen, die für das eigentliche Projekt als Abhängigkeiten benötigt werden. Um die Zeit für den Gang durch die New-Queue zu überbrücken, können diese auch lokal für das Bauen in der *chroot* bereitgestellt werden.

```

371 <UploadLocal 371>≡ (372)
function UploadLocal {
 # Called by TaskSelect
 if [${Upl} -eq 5]
 then
 # Provide for local chroot
 UploadFilesSelect
 if whiptail --title " Files Uploaded?" \
--yesno "Should the following files\n \
${UplFL2}\nhave to be uploaded?" 15 60
 then
 cd ${ProjectPath}/${OrigName}
 sudo cp ${UplFL2} /var/local/repository
 UpdateLocalRepo
 fi
 fi
}

<ChangeEntry (nicht definiert)>

```

Die folgende Funktion gibt es bereits als Shell-Skript unter `/usr/local/bin/LocalNewRepo`.

```
372 <UpdateLocalRepo 372>≡ (370)
function UpdateLocalRepo {
 # Called by UploadLocal
 cd /var/local/repository

 # Make package archives writable
 # (not only for root)
 sudo chmod o+w Packages
 sudo chmod o+w Sources
 sudo chmod o+w Packages.gz
 sudo chmod o+w Sources.gz
 sudo chmod o+w Release

 # Use apt-ftparchive to update package archives
 sudo apt-ftparchive packages . > Packages &&
 sudo apt-ftparchive sources . > Sources &&
 sudo cat Packages | gzip -c > Packages.gz &&
 sudo cat Sources | gzip -c > Sources.gz &&
 sudo apt-ftparchive release . > Release

 # Reset rights
 sudo chmod o-w Packages
 sudo chmod o-w Sources
 sudo chmod o-w Packages.gz
 sudo chmod o-w Sources.gz
 sudo chmod o-w Release
}
```

<UploadLocal 371>

In die *Chroot-Umgebung*, hier `/var/cache/pbuilder/base.cow`, wird in die Datei `/etc/apt/sources.list` folgendes eingefügt:

```
deb [trusted=yes] file:///var/local/repository ./
deb-src [trusted=yes] file:///var/local/repository ./
```



**Teil V.**

**Weitere Bestandteile des  
Skriptes**



## 44. Weitere Aufgaben

### 44.1. Neuen Branch erstellen

```
375 <CreateNewBranch 375>≡
function CreateNewBranch {
 # Called by TaskSelect

 # Creates a new branch (for backports or proposed-updates)
 DebianBranches
 whiptail --title "Recent branches" \
 --msgbox "Recent branches:\n${bl}" 15 60
 bName=""
 bName=$(whiptail --inputbox "Name of the new branch:" \
 --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
 if [${bName} != ""]
 then
 ## Create new branch in git
 git checkout -b ${bName}

 ## Change config file - make new branch to recent one
 ChangeEntry

 whiptail --title "New branch was created" \
 --msgbox "New branch ${bName} was created" 15 60
 echo "New branch ${bName} was created" >> ${log}
 Distro4Branch
 fi
}

<ParseConfig (nicht definiert)>
```

Nun wird wieder die Aufgabenauswahl aufgerufen (Kapitel 33.5, Seite 177).

## 44.2. Eingabe des Namens oder der IP eines eigenen Git-Servers

Mit dieser Funktion, die von der Aufgabenauswahl (Kapitel 33.5, Seite 177) aufgerufen werden kann, werden der Name oder die IP eines eigenen Git-Servers in die Konfigurationsdatei eingetragen. Die Eingabe eines Namens setzt eine funktionierende Namensauflösung voraus.

```
376 <OwnServer 376>≡ (377)
 function OwnServer {
 # Called by TaskSelect

 # Set name or IP of own git server
 ServerName=$(whiptail --inputbox "Name or IP-address of your git server:" \
 --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
 if [-z "${ServerName}"]
 then
 echo "Name or IP of your git server:"
 read ServerName
 fi
 if [-n "$ServerName"]
 then
 # ReplaceConfigLines needs two parameters:
 # name of the variable and new value
 ReplaceConfigLines 'ServerName' "${ServerName}"
 AddGitServer
 fi
 }

 <Ask4DebDiff (nicht definiert)>
```

Das Programmskript ruft die Funktion *ReplaceConfigLines* (Kapitel 33.2, Seite 167) auf, um den Git-Server in die Konfigurationsdatei einzutragen. Außerdem wird die Funktion *AddGitServer* aufgerufen.

### 44.3. AddGitServer

Die Funktion *AddGitServer* zeigt eine Liste der bisherigen Remote-Server an und fragt, ob der eigene Git-Server hinzugefügt werden soll.

Wird diese Frage bejaht, wird die Funktion *AddHomeServer* (Kapitel 32.4.4, Seite 149) aufgerufen.

```
377 <AddGitServer 377>≡
 function AddGitServer {
 # Called by OwnServer
 serverlist=$(git remote -v)
 if whiptail --title "Recent remote servers" \
 --yesno "${serverlist}\nAdd git remote server 'home'?" \
 --yes-button "Yes" \
 --no-button "No" 15 60
 then
 AddHomeServer
 fi
 }

 <OwnServer 376>
```



## 45. Kopf des Skriptes

### 45.1. Shebang

Am Anfang des Skriptes befinden sich die *Shebang*, Vermerke über die Autoren, die Version und die Lizenz.

```
379a <build-gbp.sh 379a>≡
 #!/usr/bin/bash

 <copyright 379b>
```

### 45.2. Copyright-Vermerk

Auf die *Shebang* folgt der Copyright-Vermerk.

```
379b <copyright 379b>≡ (379a)
 # Copyright 2019-2025 Mechtilde and Michael Stehmann <mechtilde@debian.org>
 # version 0.9.9

 # This program is free software; you can redistribute it and/or modify
 # it under the terms of the GNU General Public License as published by
 # the Free Software Foundation; either version 3 of the License, or
 # (at your option) any later version.

 # This program is distributed in the hope that it will be useful,
 # but WITHOUT ANY WARRANTY; without even the implied warranty of
 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 # GNU General Public License for more details.

 # You should have received a copy of the GNU General Public License
 # along with this program; if not, write to the Free Software
 # Foundation, Inc., 31 Milk Street, #960789 Boston,
 # MA 02196, USA.

 <Dependencies 379c>
```

### 45.3. Abhängigkeiten für das Programmskript

Sodann werden die Abhängigkeiten aufgeführt (s.a. Kapitel 19.1, Seite 57).

```
379c <Dependencies 379c>≡ (379b)
 # Dependencies: git-buildpackage, build-essential, less, pbuilder,
 # pristine-tar, sudo, unzip, cowbuilder, cowdancer, debmake, quilt,
 # locate, jq, lintian, devscripts, debhelper,
 # sbuild, schroot, debootstrap, apt-cacher-ng,
 # gradle-debian-helper, maven-debian-helper, libmaven-bundle-plugin-java

 <Header 380a>
```

## 45.4. `set -e` und Funktionsheader

Nach der *Debian-Policy*<sup>1</sup> soll jedes Skript entweder `set -e` benutzen oder den Exit-Status jedes Befehls prüfen.

`set -e` veranlasst das Programmskript zum sofortigen Beenden bei Fehlern. Dies ist dann der Fall, wenn ein Befehl oder eine Pipeline (Symbol: „|“) einen *Exit*-Status ungleich 0 zurückgibt.

Dieses Verhalten ist jedoch an einigen Stellen im Programmskript unerwünscht, beispielsweise wenn der Rückgabewert vom Programmskript abgefragt und zur Ablaufsteuerung verwandt wird. In diesen Fällen wird dieses Verhalten durch `set +e` temporär ausgeschaltet.

380a `<Header 380a>≡` (379c)

```
set -e

#####

Definitions of functions

#####

<DebugRP 380b>
```

Der Funktionsheader markiert für den Leser den Beginn der technischen Beschreibungen der Funktionen.

## 45.5. Funktion zur Fehlersuche

Die folgende Funktion zeigt einen Pfad und ermöglicht gegebenenfalls einen Ausstieg aus dem Programm.

Sie dient dem Debugging und ist normalerweise ungenutzt.

380b `<DebugRP 380b>≡` (380a)

```
function DebugRP {
 # Function to show a path and give an opportunity to exit
 # It is for debugging
 descstr=${1}
 pathstr=${2}
 if ! whiptail --title "Shows the path" \
 --yesno "${descstr}= ${pathstr}?" --yes-button "Yes" \
 --no-button "No" 15 60
 then
 exit
 fi
}

<InsertConfigLine (nicht definiert)>
```

<sup>1</sup><https://www.debian.org/doc/debian-policy/ch-files.html#scripts> [7]



**Teil VI.**

**Plugins und Skripte**



## 46. Java-Plugin

```
383a <build-gbp-java-plugin.sh 383a>≡
 #!/usr/bin/bash

 # Copyright 2019–2025 Mechtilde and Michael Stehmann <mechtilde@debian.org>
 # version 0.9.9

 # java plugin for build-gbp.sh

 # This program is free software; you can redistribute it and/or modify
 # it under the terms of the GNU General Public License as published by
 # the Free Software Foundation; either version 3 of the License, or
 # (at your option) any later version.

 # This program is distributed in the hope that it will be useful,
 # but WITHOUT ANY WARRANTY; without even the implied warranty of
 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 # GNU General Public License for more details.

 # You should have received a copy of the GNU General Public License
 # along with this program; if not, write to the Free Software
 # Foundation, Inc., 31 Milk Street, #960789 Boston,
 # MA 02196, USA.

 <Rules4Java 383b>
```

### 46.1. Anpassungen für Java-Pakete

In der Datei *debian/rules* (Kapitel 35.4.8, Seite 244) werden für das Kompilieren mit Java folgende Zeilen hinzugefügt:

```
383b <Rules4Java 383b>≡ (383a)
 function Rules4Java {
 echo "export JAVA_TOOL_OPTIONS := -Dfile.encoding=UTF8" >>\
 ${GitPath}/debian/rules
 echo -e "export JAVA_HOME := /usr/lib/jvm/default-java\n" >>\
 ${GitPath}/debian/rules
 }
 <build-gbp-java-plugin 384>
```

4. Mai 2025

Dies bedeutet, dass:

1. Immer das UTF-8 encoding genutzt wird
2. die Variable `JAVA_HOME` gesetzt ist.

Vom Skript wird lediglich eine Minimalkonfiguration der Datei `rules` bereitgestellt. Diese muss oft noch sinnvoll ergänzt werden. Für das Bauen der JAVA-Pakete werden häufig folgende Optionen benötigt.

```
JMODS := /usr/lib/jvm/default-java/jmods
JAVACMD="\$JAVA_HOME/bin/java"
```

Es kommt auch vor, dass `JDK_HOME` statt `JAVA_HOME` verwendet wird.

Diese Ergänzungen werden für die mit dem `maven`-Build-System erstellten Pakete vom `Maven-Plugin` (Kapitel 47, Seite 385) bereitgestellt.

In der Zeile mit `dh $@` können Optionen eingefügt werden.

```
--with javahelper
--buildsystem=maven
--buildsystem=ant
--buildsystem=gradle
```

384 `<build-gbp-java-plugin 384>≡`

(383b)

```
whiptail --title "Java plugin found" \
--msgbox "build-gbp-java-plugin.sh was loaded." 15 60

echo "build-gbp-java-plugin.sh was loaded." >> ${log}
```

# 47. Maven-Plugin

Das Maven-Plugin unterstützt das Bauen von Java-Paketen mit dem Java-Build-System *maven* (Kapitel 13.4.1, Seite 41).

## 47.1. Kopf des Maven-Plugins

Der Kopfteil des Plugins enthält Angaben zu den Urhebern und der Lizenz.

Außerdem ist vermerkt, welche Debian-Pakete für den Ablauf des Programmskriptes installiert sein müssen.

```
385a <build-gbp-maven-plugin.sh 385a>≡
 #!/usr/bin/bash

 # Copyright 2019-2025 Mechtilde and Michael Stehmann <mechtilde@debian.org>
 # version 0.9.9

 # maven plugin for build-gbp.sh

 # This program is free software; you can redistribute it and/or modify
 # it under the terms of the GNU General Public License as published by
 # the Free Software Foundation; either version 3 of the License, or
 # (at your option) any later version.

 # This program is distributed in the hope that it will be useful,
 # but WITHOUT ANY WARRANTY; without even the implied warranty of
 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 # GNU General Public License for more details.

 # You should have received a copy of the GNU General Public License
 # along with this program; if not, write to the Free Software
 # Foundation, Inc., 31 Milk Street, #960789 Boston,
 # MA 02196, USA.

 # Dependencies: maven-debian-helper licensecheck apt-file

 <Rules4MavenDH 398>
```

## 47.2. Mitteilung

Das Plugin-Skript teilt mit, dass es geladen wurde. Damit ist der Code des Plugins Teil des (Haupt-)Programmskriptes.

```
385b <MavenPlugin 385b>≡ (395a)
 whiptail --title "maven plugin found" \
 --msgbox "build-gbp-maven-plugin.sh was loaded." 15 60

 echo "build-gbp-maven-plugin.sh was loaded." >> ${log}
 # Next the function MakeMaven is executed by the main script.
 # This is the end, my friend
```

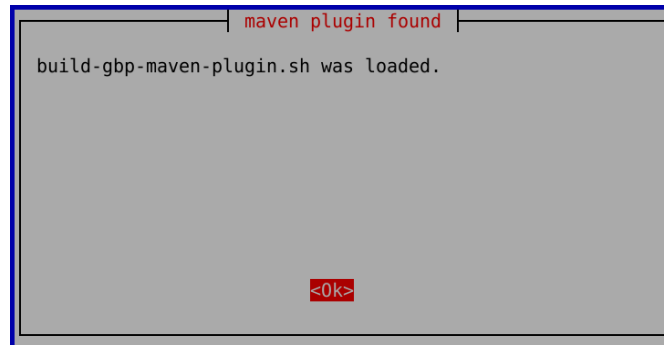


Abbildung 47.1.: Maven Plugin geladen

### 47.3. Bauen mit *Maven*

Diese Funktion wird nach dem Laden des Plugins beim Bauen einer neuen Revision von der Funktion *BuildNewVersion* des (Haupt-)Programmskriptes aufgerufen (Kapitel 35.2, Seite 227).

Die Ausführung von *mh\_make* erfolgt in einer *Chroot*-Umgebung. Wenn die *chroot* noch nicht existiert, ist sie zunächst als */srv/maven-chroot* anzulegen wie in Kapitel 19.5 (Seite 69) beschrieben.

```
386a <MakeMaven 386a>≡ (391)
 function MakeMaven {
 # Called by BuildNewRevision

 cd ${GitPath}
 IdentifyMavenChrootPath

 <MakeMaven1 392>
```

Nun wird die Funktion *IdentifyMavenChrootPath* aufgerufen. Diese Funktion prüft, ob die Variable *ChrootPath* mit einem Wert belegt ist.

```
386b <IdentifyMavenChrootPath 386b>≡ (390a)
 function IdentifyMavenChrootPath {
 # Called by MakeMaven and itself

 if [-n "${ChrootPath}"]
 then
 if ! whiptail --title "Maven chroot path " \
 --yesno "Is ${ChrootPath}\nthe right path to\n\
 maven chroot directory on the host?" \
 --yes-button "Yes" \
 --no-button "No" 15 60
 <IdentifyMavenChrootPath1 387a>
```

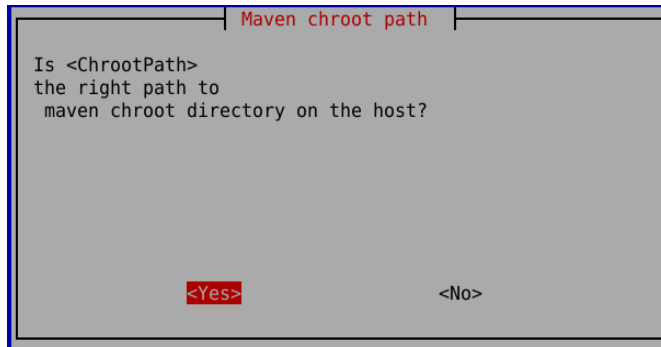


Abbildung 47.2.: Pfad zur Maven-Chroot ermitteln

Wird der korrekte Pfad zur *maven-chroot* angezeigt, geht es weiter mit der Prüfung, ob die *maven-chroot* auch an diesem Ort eingerichtet ist.

```

387a <IdentifyMavenChrootPath1 387a>≡ (386b)
 then
 OldCP=${ChrootPath}
 AskChrootPath
 fi
 else
 AskChrootPath
 fi

 <IdentifyMavenChrootPath2 388b>

387b <AskChrootPath 387b>≡ (394b)
 function AskChrootPath {
 # Called by IdentifyMavenChrootPath

 # This is the way from GitPath to /srv/maven-chroot
 ChrootPath=$(whiptail --title "Maven chroot path" \
 --inputbox "Please insert the path\nto the maven chroot directory\n\
 on the host:" \
 --nocancel 15 60 3>&2 2>&1 1>&3)
 <AskChrootPath1 388a>
 }

```

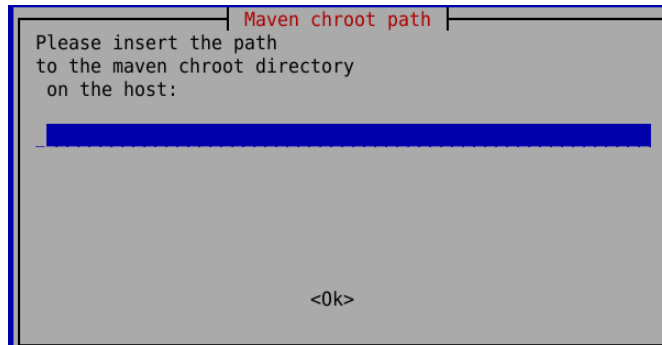


Abbildung 47.3.: Pfad zur Maven-Chroot angeben

388a `<AskChrootPath1 388a>≡` (387b)

```

if [-n "${OldCP}"]
then
 ReplaceConfigLines "ChrootPath" ${ChrootPath}
else
 echo "### Maven chroot path"
 echo "ChrootPath=${ChrootPath} >> ${ConfigPath}${OrigName}"
fi
OldCP=""
}

```

`<AskWorkspacePath 389c>`

Ist die *maven-chroot* noch nicht eingerichtet, wird sie vom Programm-Skript im angegebenen Verzeichnis eingerichtet.

388b `<IdentifyMavenChrootPath2 388b>≡` (387a)

```

Create Maven-Chroot if not exists
if ! [-d ${ChrootPath}]
then
 echo "Please enter your SUDO password!"
 sudo mkdir --parents ${ChrootPath}
fi

if ! [-d ${ChrootPath}/usr]
then
 echo "Please enter your SUDO password!"
 sudo /usr/sbin/debootstrap --arch amd64 sid \
 ${ChrootPath} http://ftp.de.debian.org/debian
 echo "The maven chroot has been created." >> ${log}
fi

```

`<IdentifyMavenChrootPath3 389a>`



Danach wird der Pfad für den Arbeitsbereich in der *maven-chroot* ermittelt.

```
389a <IdentifyMavenChrootPath3 389a>≡ (388b)
 # The workspace is /home/user
 if [-n "${Path2Workspace}"]
 then
 if ! whiptail --title "Maven chroot path " \
 --yesno "Is ${Path2Workspace}\nthe workspace in the maven chroot?" \
 --yes-button "Yes" \
 --no-button "No" 15 60 # test
 <IdentifyMavenChrootPath4 389b>
```

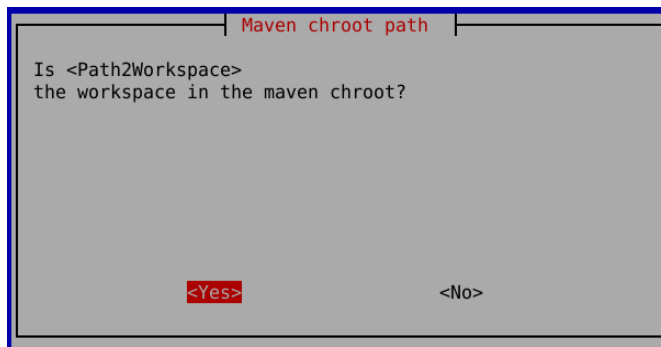


Abbildung 47.4.: Arbeitsverzeichnis in der Maven-Chroot ermitteln

Wird die Frage nach dem Arbeitsbereich innerhalb der *maven-chroot* bejaht, wird das Arbeitsverzeichnis des Projektes in den Arbeitsbereich der *maven-chroot* kopiert (Seite 392).

Sodann geht es mit der Arbeit innerhalb der *maven-chroot* weiter.

```
389b <IdentifyMavenChrootPath4 389b>≡ (389a)
 then
 OldP2W=${Path2Workspace}
 AskWorkspacePath
 fi
 else
 AskWorkspacePath
 fi
 <IdentifyMavenChrootPath5 390b>

389c <AskWorkspacePath 389c>≡ (388a)
 function AskWorkspacePath {
 # Called by IdentifyMavenChrootPath

 # This is the way to /home/user/
 Path2Workspace=$(whiptail --title "Maven chroot path" \
 --inputbox "Please insert the path\nto the workspace directory:" \
 --nocancel 15 60 3>&2 2>&1 1>&3)
 <AskWorkspacePath2 390a>
```

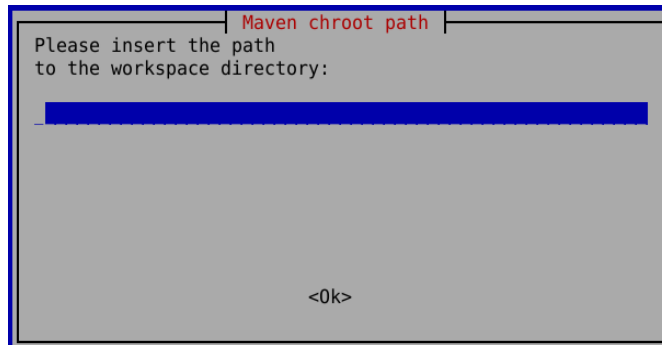


Abbildung 47.5.: Arbeitsverzeichnis in der Maven-Chroot angeben

390a `<AskWorkspacePath2 390a>≡` (389c)

```

if [-n "${OldP2W}"]
then
 ReplaceConfigLines "Path2Workspace" "${Path2Workspace}"
else
 echo "Path2Workspace=${Path2Workspace} >> ${ConfigPath}${OrigName}"
fi
OldP2W=""
}

```

`<IdentifyMavenChrootPath 386b>`

390b `<IdentifyMavenChrootPath5 390b>≡` (389b)

```

Replace / at the end
Path2Workspace=$(echo ${Path2Workspace} | sed --expression="s/\$/"/)
MavenChrootPath=${ChrootPath}${Path2Workspace}
if ! [-d ${MavenChrootPath}]
then
 whiptail --title "Error!" \
 --msgbox "${MavenChrootPath} does not exist.\n\
 It will be created now." 15 60

```

`<IdentifyMavenChrootPath7 391>`

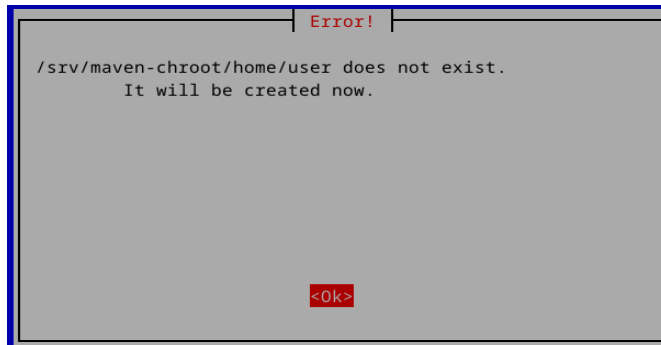


Abbildung 47.6.: Maven-Chroot existiert nicht

Existiert die *chroot* nicht, ist sie anzulegen wie in Kapitel 19.5, Seite 69 beschrieben.

```
391 <IdentifyMavenChrootPath7 391>≡ (390b)
 echo "Please enter your SUDO password!"
 sudo mkdir --parents ${MavenChrootPath}
 echo "The maven work space has been created." >> ${log}
 fi
}

<MakeMaven 386a>
```

4. Mai 2025

Das Arbeitsverzeichnis des Projektes wird in den Arbeitsbereich der *maven-chroot* kopiert. Hierzu werden *Root*-Rechte benötigt.

```
392 <MakeMaven1 392>≡ (386a)
Copy workspace to maven chroot
echo "To copy the sources into maven-chroot you need sudo rights"
sudo cp --archive ${GitPath} ${MavenChrootPath}

Because mh_make has to run in the directory, where pom.xml is
echo "pom.xml is here:"
find . -name 'pom.xml' -print
echo "Please switch to the Maven chroot in another terminal."
echo
echo "Use these commands:"
echo
echo "# sudo mount -o bind /proc "${ChrootPath}"/proc"
echo "# sudo mount devpts /dev/pts -t devpts"
echo
echo "sudo LANG=C chroot "${ChrootPath}" /usr/bin/bash"
echo "cd "${Path2Workspace}"/"${SourceName}"
echo "try: apt install maven-debian-helper"
echo "apt update && apt upgrade"
echo "mh_make --verbose "${SourceName}"
echo
echo "Then follow the questions of mh_make"
echo
echo "You can leave the chroot with 'exit'."
echo
echo "After finishing press return to go on!"
read a

#--run-tests=true --javadoc=true --verbose
#mh_make --package=${SourceName} --bin-package=${PackName} \
#--run-tests=false --javadoc=false --verbose
#if [$? -ne 0]
#then
echo "mh_make failed!"
exit
#else
echo "mh_make has been executed for "${PackName} >> ${log}
#fi

cp --recursive --update ${MavenChrootPath}/${SourceName}/debian \
${GitPath}
```

<MakeMaven5 394a>

Beim Ablauf des Programmes *mh\_make* sind folgende Fragen zu beantworten. Dieses Programm selber ist auch ein Shell-Skript.

**Environment variable DEBLICENSE not set, using Apache-2.0 by default** Es wird standardmäßig die Lizenz Apache-2.0 verwendet, wenn die Umgebungsvariable *DEBLICENSE* nicht gesetzt ist. In der Variablen *DEBLICENSE* wird die Lizenz für die Dateien in *debian/* hinterlegt.

**Checking that apt-file is installed and has been configured... [ok]** Es wird geprüft, ob das Paket *apt-file* installiert ist. Dieses Paket ist nur als *empfohlen* markiert.

**Checking that licensecheck is installed... [ok]** Prüfung, ob das Paket *licensecheck* installiert ist. Dieses Paket ist ebenfalls nur als *empfohlen* markiert.

**Solving dependencies for package <PaketName>**

**Analysing pom.xml...**

**Resolving com.jcabi:jcabi:pom:1.21 of scope runtime...**

**The parent POM cannot be found in the Maven repository for Debian.** Es ist der Regelfall, dass die Abhängigkeit für das Elternelement nicht gefunden wird. Daher wird die Frage bejaht, dass dies ignoriert werden kann. Diese Information wird dann als *-no-parent* in die Datei *<PaketName>.poms* geschrieben.

**Enter the upstream version for the package.** Hier wird die zu bauende Versionsnummer des Upstream-codes abgefragt und in der Regel mit *Enter* bestätigt. Andernfalls wird die korrekte Versionsnummer eingegeben.

**Version of com.jcabi:jcabi-aspects is 0.22.6** Es wird nochmal die Versionsnummer angezeigt.

**Choose how the version will be transformed:** Nun wird eine Auswahl angezeigt, wie diese Versionsnummer umgewandelt werden soll.

**0 - Replace all versions starting by 0. with 0.x** Es sollen alle Versionsbezeichnungen ersetzt werden.

**(1) - Change the version to the symbolic 'debian' version** Die Standardangabe wird mit eckigen Klammern (Hier sind es aus satztechnischen Gründen runde Klammern) gekennzeichnet. Diese kann mit *Enter* bestätigt werden.

**2 - Keep the version**

**3 - Custom rule**

Andernfalls wird nun die gewünschte Ziffer eingegeben.

**Inform mh\_make that dependencies of type jar which may match this library should be transformed into bundles**

Soll *mh\_make* mitgeteilt werden, dass die passenden Abhängigkeiten vom Typ *jar* in Bundles umgewandelt werden. Standardmäßig wird diese Frage bejaht.

**Resolving com.jcabi:jcabi-log:jar:0.17 of scope runtime... [ check dependency with bundle type ]**

**In pom.xml: This dependency cannot be found in the Debian Maven repository. Ignore this dependency? com.jcabi:jcabi-log:jar:0.17 [ y/N ]**

Diese Meldung kommt, wenn die benötigte Abhängigkeit nicht auf der Arbeitsmaschine installiert ist.

- `>dpkg -search /usr/share/maven-repo/com/jcabi/jcabi-log/*/*` dpkg failed to execute successfully
- `>apt-file search /usr/share/maven-repo/com/jcabi/jcabi-log`  
Found /usr/share/maven-repo/com/jcabi/jcabi-log/0.18.1/jcabi-log-0.18.1.pom  
in libjcabi-log-java  
Found /usr/share/maven-repo/com/jcabi/jcabi-log/0.19.0/jcabi-log-0.19.0.pom  
in libjcabi-log-java  
Found /usr/share/maven-repo/com/jcabi/jcabi-log/debian/jcabi-log-debian.pom  
in libjcabi-log-java
- `>dpkg -search /usr/share/java/jcabi-log.jar` dpkg failed to execute successfully

- `>apt-file search /usr/share/java/jcabi-log.jar` Found libjcabi-log-java [error] Package libjcabi-log-java does not contain Maven dependency com.jcabi:jcabi-log:jar:0.17 but there seem to be a match If the package contains already Maven artifacts but the names don't match, try to enter a substitution rule of the form `s/groupId/newGroupId/ s/artifactId/newArtifactId/ jar s/version/newVersion/` here: `>`

```
394a <MakeMaven5 394a>≡ (392)
 if [-w debian/maven.rules]
 then
 echo "#[groupId] [artifactId] [type] [version] [classifier] [scope]" \
 >> debian/maven.rules
 fi

 ShowMaven

 <MakeMaven6 395a>
```

## 47.4. Maven-Dateien bearbeiten

Pakete, die mit *Maven* (*mvn*) gebaut werden, benötigen dazu weitere Dateien im Verzeichnis *debian/*. Diese werden im Folgenden aufgelistet.

Die folgenden Daten wurden in der *maven-chroot* erstellt:

- `<PackName.poms>`
- `maven.cleanIgnoreRules`
- `maven.properties`
- `maven.rules`
- `maven.ignoreRules`
- `maven.publishedRules`

Nun können auch diese Dateien angezeigt und editiert werden.

```
394b <ShowMaven 394b>≡ (398)
 function ShowMaven {
 # Called by DisplayDebianFiles MakeMaven

 mavenl=$(ls ${GitPath}/debian/ | grep 'maven')
 for element in ${mavenl[*]}
 do
 nano --linenumbers --mouse --softwrap debian/${element}
 done

 pomsl=$(ls ${GitPath}/debian/ | grep ${PackName})
 for element in ${pomsl[*]}
 do
 nano --linenumbers --mouse --softwrap debian/${element}
 done
 }

 <AskChrootPath 387b>
```

395a  $\langle$ MakeMaven6 395a $\rangle$  $\equiv$ 

(394a)

```

Patch for mh_make bug
str4standardsversion="4.7.1"
cd ${GitPath}/debian/
less --LINE-NUMBERS control

sed --in-place --expression=\
"s/Standards-Version: 4.4.1/Standards-Version: ${str4standardsversion}/" \
control
'a' means append. The string after the 'a' will be appended
to the sting before the 'a'.
sed --in-place \
--expression="/Standards-Version: ${str4standardsversion}/ \
a Rules-Requires-Root: no" \
control
sed --in-place --expression=\
"s/debhelper-compat (=12)/debhelper-compat ${str4versiondebhelpers}/" \
control

cd ${GitPath}
whiptail --title "Check debian/control" \
--msgbox "Please check the control file another time!" 15 60
less --LINE-NUMBERS ${GitPath}/debian/control
}

 \langle MavenPlugin 385b \rangle

```

#### 47.4.1. maven.rules

Diese Datei ist wie folgt aufgebaut:

395b  $\langle$ maven-rules.header 395b $\rangle$  $\equiv$   
 #[groupID] [artifactID] [type] [version] [classifier] [scope]

#### 47.4.2. maven.ignoreRules

Diese Datei ist wie die Datei *maven.rules* aufgebaut.

395c  $\langle$ maven-ignoreRules.header 395c $\rangle$  $\equiv$   
 #[groupID] [artifactID] [type] [version] [classifier] [scope]

Artefakte, die in der Datei *maven.rules* aufgenommen werden, sind hier gegebenenfalls zu löschen

### 47.4.3. maven.properties

Die Datei *maven.properties* kann verwendet werden, um Werte von Variablen innerhalb der *pom.xml*-Dateien zu überschreiben. Der häufigste Anwendungsfall ist die Deaktivierung oder Aktivierung der Tests mit *maven.test.skip=true*

Weitere Optionen sind die Änderung der Quell- (*maven.compiler.source=8*) bzw. Zielebene (*maven.compiler.target=8*). Dabei bezieht sich der Wert (hier: 8) auf die java-Mindestversion. Dieser Eintrag wird besonders dann benötigt, wenn beim Kompilieren eine Meldung wie die folgende ausgegeben wird: *use -source 8 or higher to enable ...*

Außerdem kann hier die verwendete Dateikodierung (*project.build.sourceEncoding=UTF-8*) eingestellt werden.

```
396 <maven.properties 396>≡
 # Include here properties to pass to Maven during the build.
 # For example:
 # maven.test.skip=true
 # project.build.sourceEncoding=UTF-8
 # maven.compiler.source=8
 # maven.compiler.target=8
 # To skip test - missing dependencies
 maven.test.skip=true
```



Häufig werden die Tests deaktiviert, um nicht zusätzliche Abhängigkeiten paketieren zu müssen. In diesen Fällen sind diese Abhängigkeiten auch in der Datei *pom.xml* mit einem Kommentarzeichen zu versehen oder zu entfernen.

Solche Änderungen am Upstream-Code können mit dem Programmskript vorgenommen werden (Kapitel 36, Seite 253).

#### 47.4.4. Paketname.poms

In der Datei *<Paketname>.poms* werden die Optionen zu den jeweiligen *pom.xml*-Dateien hinterlegt.

Die Datei enthält dann 2 Spalten. In der ersten Spalte steht die *pom.xml* mit dem dazugehörigen Pfad und in der zweiten Spalte die dazugehörigen Optionen, wie z.B.

```
List of POM files for the package
Format of this file is:
<path to pom file> [option]*
where option can be:
--ignore: ignore this POM and its artifact if any
--ignore-pom: don't install the POM. To use on POM files that are
created temporarily for certain artifacts such as Javadoc jars.
[mh_install, mh_installpoms]
--no-parent: remove the <parent> tag from the POM
--package=<package>: an alternative package to use when installing
this POM and its artifact
--has-package-version: to indicate that the original version of
the POM is the same as the upstream part of the version for the
package.
--keep-elements=<elem1,elem2>: a list of XML elements to keep in the
POM during a clean operation with mh_cleanpom or mh_installpom
--artifact=<path>: path to the build artifact associated with this
POM, it will be installed when using the command mh_install.
[mh_install]
--java-lib: install the jar into /usr/share/java to comply with
Debian packaging guidelines
--usj-name=<name>: name to use when installing the library in
/usr/share/java
--usj-version=<version>: version to use when installing the library
in /usr/share/java
--no-usj-versionless: don't install the versionless link in
/usr/share/java
--dest-jar=<path>: the destination for the real jar.
It will be installed with mh_install. [mh_install]
--classifier=<classifier>: Optional, the classifier for the jar.
Empty by default.
--site-xml=<location>: Optional, the location for site.xml if it
needsto be installed.
Empty by default. [mh_install]
#
pom.xml --no-parent --has-package-version
```

Dies bedeutet, dass das *<parent>*-Tag beim Bauen aus der POM entfernt wird. Die Option *-has-package-version* gibt an, dass die Originalversion der POM dieselbe sei, wie der Upstream-Teil der Paketversion.

#### 47.4.5. README.source

Information about jcabi-aspects  
-----

This package was debianized using the `mh_make` command from the `maven-debian-helper` package.

The build system uses Maven but prevents it from downloading anything from the Internet, making the build compliant with the Debian policy.

### 47.5. *debian/rules* - Ergänzungen für Java-Pakete mit *Maven*

In der Funktion *Rules4MavenDH* wird die Datei *debian/rules* um die Angabe des Build-Systems ergänzt.

```
398 <Rules4MavenDH 398>≡ (385a)
 function Rules4MavenDH {
 # Called by DebianRulesTemplates

 sed --in-place \
 --expression="s/dh \$/dh \$/ --buildsystem=maven/" \
 ${GitPath}/debian/rules

 }
 <ShowMaven 394b>
```

## 48. Web-Extension-Plugin

Das nachstehend dargestellte Plugin erfüllt die besonderen Anforderungen beim Bauen von Mozilla-Erweiterungen als Debian-Pakete (Kapitel 14, Seite 47).

### 48.1. Kopf des Webext-Plugins

Der Kopfteil des Plugins enthält Angaben zu den Urhebern und der Lizenz.

Außerdem ist vermerkt, welche Debian-Pakete für den Ablauf des Programmskriptes installiert sein müssen.

```
399 <build-gbp-webext-plugin.sh 399>≡
 #!/usr/bin/bash

 # Copyright 2020–2025 Mechtilde and Michael Stehmann <mechtilde@debian.org>
 # version 0.9.9

 # webext plugin for build-gbp.sh

 # This program is free software; you can redistribute it and/or modify
 # it under the terms of the GNU General Public License as published by
 # the Free Software Foundation; either version 3 of the License, or
 # (at your option) any later version.

 # This program is distributed in the hope that it will be useful,
 # but WITHOUT ANY WARRANTY; without even the implied warranty of
 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 # GNU General Public License for more details.

 # You should have received a copy of the GNU General Public License
 # along with this program; if not, write to the Free Software
 # Foundation, Inc., 31 Milk Street, #960789 Boston,
 # MA 02196, USA.

 <IdentifyWebextId 400>
```

## 48.2. Erstellen der *webext\*.xpi*-Dateien in *debian/*

### 48.2.1. Ermittlung des Namens der *\*.xpi*-Datei

Die zu paketierende Erweiterung muss unter dem Namen der sich aus der Bezeichnung (id) in der Datei *manifest.json* und der Endung *.xpi* zusammensetzt, abgelegt werden.

Die Bezeichnung wird daher wie folgt ermittelt.

```

400 <IdentifyWebextId 400>≡ (399)
 function IdentifyWebextId {
 # Called by WebextRulesDH WebextInstall

 if [-z ${webextID}] && [-f ${GitPath}/manifest.json]
 then
 webextID=$(grep 'id:' ${GitPath}/manifest.json)
 webextID=$(echo ${webextID} | sed --expression='s/^\s"id\": \s//')
 webextID=$(echo ${webextID} | sed --expression='s/\s,\s*//')

 echo -e "Notice from Webext-Plugin: WebextID = "${webextID} >> ${log}
 fi

 if [-z ${webextID}]
 then
 webextID="PLEASE_REPLACE_WITH_ID"
 whiptail --title "ID not found" \
 --msgbox "Didn't find ID or manifest.json in ${GitPath}" 15 60
 fi
 }

 <webext-rules 401a>

```

### 48.2.2. *debian/rules* - Ergänzungen für Mozilla-AddOns

Bei der *debian/rules*-Datei für die Webextension ist es sinnvoll, die zu installierenden Verzeichnisse bzw. Dateien dort aufzuführen und einer Variablen zuzuweisen. Dies erfolgt nach dem Muster `<Package>_FILES = <List of files to include>`

Die Liste `<Package>_FILES` wird vom Plugin-Skript automatisch erstellt. Sie ist auf Vollständigkeit und Richtigkeit zu überprüfen. Die auszuschließenden Dateien sind manuell einzupflegen.

Da die Anlage der Datei *debian/rules* einmalig erfolgt, sind beim Paketieren neuer Versionen die beiden Listen sorgfältig zu prüfen und gegebenenfalls zu anzupassen.

401a `<webext-rules 401a>≡` (400)

```
function WebextRules {
 # Called by DebianRulesTemplate

 # These strings will be added to str4rules

 Package=$(echo ${SourceName} | tr "a-z" "A-Z")
 echo -e "${Package}_FILES = \" \" >> ${GitPath}/debian/rules

 PackageL=$(ls ${GitPath})
 PackageA=(${PackageL})

 for element in ${PackageA[*]}
 do
 if ["${element}" != "debian"]
 then
 echo -e "${element}" \" \" >> ${GitPath}/debian/rules
 fi
 done
 echo "\$(NULL)" >> ${GitPath}/debian/rules
```

`<webext-rules5 401b>`

401b `<webext-rules5 401b>≡` (401a)

```
echo -e "\n Uncomment the following lines \n"
echo "if there are files to exclude and add them."
echo -e "\n# "${Package}_FILES_EXCLUDE = \" \" \
>> ${GitPath}/debian/rules
echo -e "# \$(NULL)\n" >> ${GitPath}/debian/rules
}
```

`<webext-rules-dh 402a>`

4. Mai 2025

Die folgenden Zeilen werden vom Programmskript der Variablen *str4rulesdh* hinzugefügt.

```
402a <webext-rules-dh 402a>≡ (401b)
function WebextRulesDH {
 # Called by DebianRulesTemplate

 IdentifyWebextId

 DHCleanStr="override_dh_clean:\n\tdh_clean\n\trm -rf debian/build\n"

 DHAutoBuildIndepStr1="override_dh_auto_build-indep:"
 DHAutoBuildIndepStr2="\tmkdir \${CURDIR}/debian/build && \\"
 Str3B="\tzip --recurse-paths "
 DHAutoBuildIndepStr3=${Str3B}\${CURDIR}/debian/build/"${webextID} ".xpi \\"
 DHAutoBuildIndepStr4="\t \${Package}_FILES) \\"
 DHAutoBuildIndepStr5="# \t --exclude \${Package}_FILES_EXCLUDE)"
 DHAutoBuildIndepStr6="\tdh_auto_build\n"

 echo -e ${DHCleanStr} >> ${GitPath}/debian/rules
 echo -e ${DHAutoBuildIndepStr1} >> ${GitPath}/debian/rules
 echo -e ${DHAutoBuildIndepStr2} >> ${GitPath}/debian/rules
 echo -e ${DHAutoBuildIndepStr3} >> ${GitPath}/debian/rules
 echo -e "${DHAutoBuildIndepStr4}" >> ${GitPath}/debian/rules
 echo -e "${DHAutoBuildIndepStr5}" >> ${GitPath}/debian/rules
 echo -e ${DHAutoBuildIndepStr6} >> ${GitPath}/debian/rules
}

<WebextControl 402b>
```

### 48.2.3. *debian/control* - Ergänzungen für Mozilla-AddOns

```
402b <WebextControl 402b>≡ (402a)
function WebextControl {
 # Called by DebianControlTemplate
 # TB specific, for FF 'web'
 sed --in-place \
 --expression="s/Section: /Section: mail/" ${GitPath}/debian/control

 # 'a' means append. The string after the 'a' will be appended
 # to the sting before the 'a'.
 # @X escapes the space at the beginning of the appended line.
 # It will be removed later.
 sed --in-place \
 --expression="/Build-Depends: debhelper-compat ${str4versiondebhelpers}/ \
a @X , zip" \
 ${GitPath}/debian/control
 # TB specific, for FF 'firefox-esr (>= 91.4)'
 sed --in-place \
 --expression="/Depends: \${misc:Depends}/ \
a @X , thunderbird (>= 1:128.3)" \
 ${GitPath}/debian/control
 sed --in-place --expression="s/^@X//g" ${GitPath}/debian/control
}

<WebextInstall 403a>
```

#### 48.2.4. *debian/webext-\*.install*

Für Mozilla-Erweiterungen ist zwingend der folgende Eintrag erforderlich:

```
debian/build/<manifest-id>.xpi /usr/share/webext
```

403a `<WebextInstall 403a>≡` (402b)

```
function WebextInstall {
 # Called by DisplayDebianFiles
 IdentifyWebextId
 InstallStr="debian/build/"${webextID} ".xpi\tusr/share/webext"
 echo -e ${InstallStr} >> ${GitPath}/debian/${PackName}.install
}
```

`<WebextDocs 403b>`

#### 48.2.5. *debian/webext-\*.docs*

403b `<WebextDocs 403b>≡` (403a)

```
function WebextDocs {
 # Called by
 echo "Still empty"
}
```

`<WebextLinks 403c>`

#### 48.2.6. *debian/webext-links-tb*

```
\# Source Target
/usr/share/webext/<manifest.id>.xpi /usr/lib/thunderbird/extensions/<manifest.id>.xpi
```

403c `<WebextLinks 403c>≡` (403b)

```
function WebextLinksTB {
 # Called by DisplayDebianFiles
 IdentifyWebextId
 SourceStr="/usr/share/webext/"${webextID} ".xpi\t"
 TargetStr="/usr/lib/thunderbird/extensions/"${webextID} ".xpi"
 echo -e ${SourceStr}${TargetStr} >> \
 ${GitPath}/debian/${PackName}.links
}
```

`<webext-plugin-end 403d>`

#### 48.2.7. Meldung: Webext-Plugin geladen

Nun kommt der Schluss dieses Webext-Plugins.

403d `<webext-plugin-end 403d>≡` (403c)

```
whiptail --title "webext plugin found" \
--msgbox "build-gbp-webext-plugin.sh was loaded." 15 60
This is the end, my friend
```

4. Mai 2025

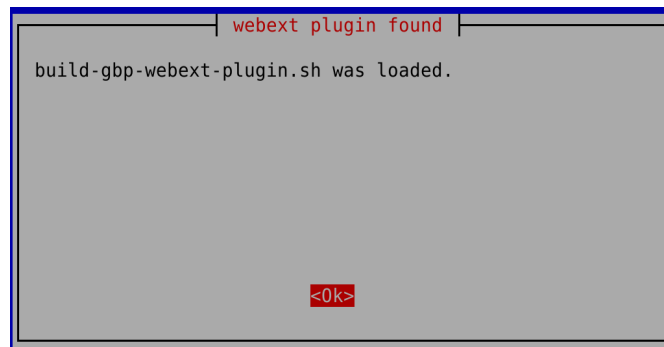


Abbildung 48.1.: Webext-Plugin wurde geladen



## 49. Python-Plugin

Das nachstehend dargestellte Plugin erfüllt die besonderen Anforderungen beim Bauen von Paketen, die in der Programmiersprache Python geschrieben sind. (Kapitel 15, Seite 49).

```
405a <build-gbp-python-plugin.sh 405a>≡
#!/usr/bin/bash

Copyright 2020-2025 Mechtilde and Michael Stehmann <mechtilde@debian.org>
version 0.9.9

python plugin for build-gbp.sh

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 31 Milk Street, #960789 Boston,
MA 02196, USA.

<python-rulesDH 406b>
```

### 49.0.1. Meldung: Python-Plugin geladen

Nun kommt der Schluss dieses Plugin-Plugins. Es gibt bekannt, dass es geladen wurde.

```
405b <IdentifyPythonId 405b>≡ (407b)
whiptail --title "python plugin found" \
--msgbox "build-gbp-python-plugin.sh was loaded." 15 60

echo "build-gbp-python-plugin.sh was loaded." >> ${log}

<python-plugin-end 407c>
```

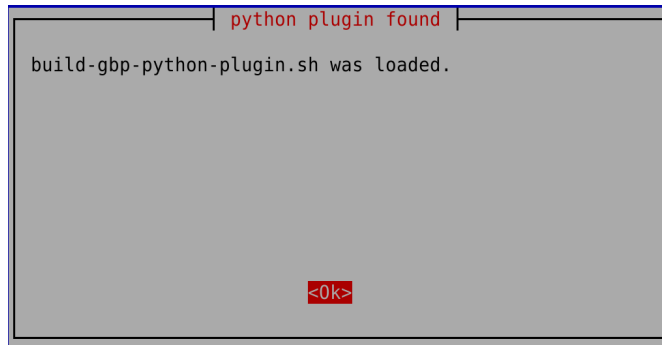


Abbildung 49.1.: Python-Plugin geladen

## 49.1. Anpassungen für Python-Pakete

Hier werden die Besonderheiten für die Python-Paketierung in der Datei *debian/rules* abgebildet.

```
406a <python-rules 406a>≡ (406b)
 function PythonRules {
 # Called by DebianRulesTemplate

 # These strings will be added to str4rules
 echo -e "export PYBUILD_NAME=\"${SourceName}\"\\n" >> ${GitPath}/Debian/rules
 echo -e "export PYBUILD_SYSTEM=distutils\\n" >> ${GitPath}/Debian/rules
 }

 <python-control 407a>

406b <python-rulesDH 406b>≡ (405a)
 function PythonRulesDH {
 # Called by DebianRulesTemplate

 sed --in-place \
 --expression="s/dh \${@}/dh \${@} --with python3 --buildsystem=pybuild/" \
 ${GitPath}/debian/rules
 }

 <python-rules 406a>
```

## 49.2. *debian/control* - Ergänzung für Python-Pakete

Hier werden die Besonderheiten für die Python-Paketierung in der Datei *debian/control* abgebildet.

```
407a <python-control 407a>≡ (406a)
function PythonControl {
 # Called by DebianControlTemplate

 # Recent Python version
 PythonVersion=$(py3versions --installed)

 sed --in-place \
 --expression="s/Section:/Section: python/" ${GitPath}/debian/control

 sed --in-place \
 --expression="/Build-Depends: debhelper-compat ${str4versiondebhelpers}/ \
a , dh-python@X , python3-all@X , python3-setuptools@X\
#X-Python3-Version: >=${PythonVersion}" ${GitPath}/debian/control
```

<python-control1 407b>

Für Tests werden auch die Pakete *python3-distutils* und *python3-distutils-extra* als Build-Abhängigkeit benötigt.

```
407b <python-control1 407b>≡ (407a)

 sed --in-place \
 --expression="/Depends: \${misc:Depends}/ \
a @X , \${python3:Depends}" \
 ${GitPath}/debian/control
```

```
 sed --in-place --expression="s/@X/\n/g" ${GitPath}/debian/control
```

}

<IdentifyPythonId 405b>

```
407c <python-plugin-end 407c>≡ (405b)

 # This is the end, my friend
```



# 50. Skripte

## 50.1. Anlage eines Projektes im Java-Team

Die Anlage eines Projektes im Java-Team, einschließlich der Erlangung eines Zugangstokens, welches der Variablen `SALSA_TOKEN` im nachfolgenden Skript zuzuweisen ist, wird in Kapitel 21.3 (Seite 76) beschrieben.

```
409a <setup-salsa-repository 409a>≡
#!/usr/bin/bash
#
Setup a new Git repository on Salsa
#
This script uses the GitLab REST API and requires an access token.
The token is obtained from the GitLab profile page -> Access Tokens
(https://salsa.debian.org/profile/personal_access_tokens).
The token is in the environment variable SALSA_TOKEN
or has to be sourced from the ~/.salsarc file and assigned
to the SALSA_TOKEN variable:
#
This is the Token for jollyday-java
SALSA_TOKEN="KyuzRyWTTxfddcGcypHn"
#

set -eu
<setup-salsa-repository3 409b>

Das Paket jq muss auf der lokalen Maschine installiert werden.

409b <setup-salsa-repository3 409b>≡ (409a)

if ! which jq >/dev/null
then
 echo "You need to apt install jq" >&2
 exit 1
fi

<setup-salsa-repository5 410a>
```

4. Mai 2025

Der Aufruf erfolgt mit der Angabe des *SourceName*.

```
410a <setup-salsa-repository5 410a>≡ (409b)
if [-z "$1"]; then
 echo "Usage: ./setup-salsa-repository <packagename>"
 exit 1;
fi

check_return_code() {
 if [$? -ne 0]; then
 echo
 echo "Something went wrong!"
 exit 1
 fi
}

test -n "$SALSA_TOKEN" || . ~/.salsarc

PACKAGE=$1

SALSA_URL="https://salsa.debian.org/api/v4"
SALSA_GROUP=java-team
SALSA_GROUP_ID=2588
```

<setup-salsa-repository8 410b>

Nun wird das Repository auf *salsa.debian.org* angelegt.

```
410b <setup-salsa-repository8 410b>≡ (410a)

echo "Creating the ${PACKAGE} repository..."

RESPONSE=$(curl -s "$SALSA_URL/projects?private_token=$SALSA_TOKEN" \
 --data "path=${PACKAGE}&namespace_id=\
 $SALSA_GROUP_ID&visibility=public&issues_enabled=false&snippets_enabled=false&wiki_enabled=false&jobs_enabled=false&printing_merge_request_link_enabled=false")

echo $RESPONSE | jq --exit-status .id > /dev/null
check_return_code

PROJECT_ID=$(echo $RESPONSE | jq '.id')
<setup-salsa-repository10 411>
```

Nun wird das ausstehende BTS-Tag-Hook konfiguriert.

```

411 <setup-salsa-repository10 411>≡ (410b)

echo "Configuring the BTS tag pending hook..."

TAGPENDING_URL="https://webhook.salsa.debian.org/tagpending/$PACKAGE"
curl --silent --output /dev/null -XPOST --header "PRIVATE-TOKEN: $SALSA_TOKEN" \
 $SALSA_URL/projects/$PROJECT_ID/hooks \
 --data "url=$TAGPENDING_URL&push_events=1&enable_ssl_verification=1"
check_return_code

echo "Configuring the KGB hook..."

KGB_URL="http://kgb.debian.net:9418/webhook/?channel=debian-java\
%26network=oftc%26private=1%26use_color=1%26use_irc_notices=1%26squash_threshold=20"
curl --silent --output /dev/null -XPOST --header "PRIVATE-TOKEN: $SALSA_TOKEN" \
 $SALSA_URL/projects/$PROJECT_ID/hooks \
 --data "url=\
 $KGB_URL&push_events=yes&issues_events=yes&\
 merge_requests_events=yes&tag_push_events=\
 yes¬e_events=yes&job_events=yes&pipeline_events=yes&\
 wiki_events=yes&enable_ssl_verification=yes"
check_return_code

echo "Configuring email notification on push..."

curl --silent --output /dev/null -XPUT --header "PRIVATE-TOKEN: $SALSA_TOKEN" \
 $SALSA_URL/projects/$PROJECT_ID/services/emails-on-push \
 --data "recipients=pkg-java-commits@lists.alioth.debian.org \
 dispatch@tracker.debian.org"
check_return_code

echo
echo "Done! The repository is located at ${SALSA_URL%/api*}/$SALSA_GROUP/$PACKAGE"

```

## 50.2. Skript zum Extrahieren der Dokumentation im PDF- und Epub-Format

Dieses Skript dient dazu, lesbare Dokumente zu erstellen.

### 50.2.1. Abhängigkeiten

Folgende Pakete müssen installiert sein, damit das Skript lauffähig ist.

- noweb
- texlive
- texlive-latex-extra
- texlive-extra-utils
- biber
- texlive-lang-japanese (für die Datei ifptex.sty)
- tidy (für das EBook)
- texlive-lang-german - für die deutschsprachige Dokumentation
- texlive-lang-english - für die englische Übersetzung
- texlive-lang-french - für die französische Übersetzung

### 50.2.2. Ablauf

Zunächst werden mit dem Skript aus den \*.nw-Dokumenten \*.tex-Dokumente erzeugt.

Das folgende Skript kann mit

```
notangle -Rcreate-book.sh Part6.nw > create-book.sh &&
t=$(date +%c)
echo "#generated on $t" >> create-book.sh
chmod ugo+x create-book.sh
```

aus der Datei *Part6.nw* extrahiert werden.

```
412 <create-book.sh 412>≡
#!/usr/bin/bash

#set -e

BasePath=$(pwd)

LANGS="en_US"

noweave -index -delay BuildWithGBP.nw > BuildWithGBP.tex # contains preamble
noweave -index -delay Title.nw > GBP-Title.tex
noweave -index -delay Part1.nw > GBP-Part1.tex
noweave -index -delay Part2.nw > GBP-Part2.tex
noweave -index -delay Part3.nw > GBP-Part3.tex
noweave -index -delay Part4.nw > GBP-Part4.tex
noweave -index -delay Part5.nw > GBP-Part5.tex
noweave -index -delay Part6.nw > GBP-Part6.tex

#noweave -filter l2h -index -html BuildWithGBP.nw | htmlltoc > BuildWithGBP.html

<create-book.sh1 413a>
```



Diese wird dann zu \*.pdf- und \*.epub-Dokumente umgewandelt. Das Stichwortverzeichnis darf nur einmal erstellt werden und zwar nach der ersten Ausführung von *pdflatex*. Mehrfache Ausführung erzeugt zu hohe Seitenzahlen.

```
413a <create-book.sh1 413a>≡ (412)
if [-f BuildWithGBP.aux]
then
rm BuildWithGBP.aux
fi

for ((i=4; i>0; i--))
do
echo $i
pdflatex -shell-escape BuildWithGBP.tex

Create Index only one time
Otherwise you get wrong page numbers
if [i=4]
then
makeindex BuildWithGBP
fi

Create Bibliography
biber BuildWithGBP
done
```

<create-book.sh5 413b>

Das EPUB-Format ist ein gezipptes XHTML, angereichert mit Metadaten.

```
413b <create-book.sh5 413b>≡ (413a)
tex4ebook -f mobi BuildWithGBP.tex
tex4ebook -f epub BuildWithGBP.tex ../

if ls | grep --quiet '\.html'
then
rm *.html
fi
```

<create-book.sh6 413c>

```
413c <create-book.sh6 413c>≡ (413b)

For the first translation

for lang in ${LANGS}
do
cd translation/${lang}/target

#sed --in-place --expression 's/^%+-.*/' GBP-Part3.tex
sed --in-place --expression ':a;N;$!ba;s/_ \n\n%/_/' GBP-Part3.tex
```

<create-book.sh7 414>

4. Mai 2025

Mit dem regulären Ausdruck werden folgende Anpassungen in der Datei *GBP-Part3.tex* vorgenommen.

Das *\$-Zeichen* steht für das Zeilenende.

```
414 <create-book.sh7 414>≡ (413c)

 if [-f BuildWithGBP.aux]
 then
 rm BuildWithGBP.aux
 fi

 for ((i=4; i>0; i--))
 do
 pdflatex -shell-escape ./BuildWithGBP.tex
 if [i=4]
 then
 makeindex BuildWithGBP
 fi

 # Create Bibliography
 biber BuildWithGBP
 done
 cd ${BasePath}
done

Remove auxillary *.html files which are needed for epub
#if ls | grep --quiet '\.html'
#then
rm *.html
#fi
```

## 50.3. Skript zum Extrahieren der Skripte

Das folgende Skript kann mit

```
notangle -Rcreate-buildscript.sh Part6.nw > create-buildscript.sh &&
t=$(date +%c)
echo "#generated on $t" >> create-buildscript.sh
chmod ugo+x create-buildscript.sh
```

aus der Datei *Part6.nw* extrahiert werden.

```
415 <build-script 415>≡
#!/bin/bash

set -e

scripts="
 build-gbp.sh
 build-gbp-java-plugin.sh
 build-gbp-maven-plugin.sh
 build-gbp-webext-plugin.sh
 build-gbp-python-plugin.sh
"

cat Title.nw Part1.nw Part2.nw Part3.nw Part4.nw Part5.nw Part6.nw > BuildWithGBPg.nw
notangle -Rbuild-gbp.sh BuildWithGBPg.nw > build-gbp.sh &&
#t=$(date +%c)
For building reproducible
t=$(date --utc --date="@${SOURCE_DATE_EPOCH:-$(date +%s)}" -R)

function build_script()
{
 notangle -R"$script" BuildWithGBPg.nw > "$script"
 sed --in-place \
 --expression='s/This is the end, my friend[[:cntrl:]]*/This is the end, my friend/' \
 "$1"
 echo "#generated on $t" >> "$1"
 chmod ugo+x "$1"
 bash -n "$1"
}

for script in $scripts; do
 build_script "$script"
done

Remove auxillary file BuildWithGBPg.nw
rm BuildWithGBPg.nw
```

4. Mai 2025

Mittels des Befehls *noweave* werden die deutschsprachigen *\*.nw*-Dateien in *\*.tex*-Dateien umgewandelt.

```
416a <CreateTexFromNW 416a>≡
function CreateTexFromNW {
 # Called by TaskSelect
 # Create *.tex files from *.nw files

 cd ${SRCDIR}

 noweave -index -delay BuildWithGBP.nw > BuildWithGBP.tex # contains preamble
 noweave -index -delay Title.nw > GBP-Title.tex
 noweave -index -delay Part1.nw > GBP-Part1.tex
 noweave -index -delay Part2.nw > GBP-Part2.tex
 noweave -index -delay Part3.nw > GBP-Part3.tex
 noweave -index -delay Part4.nw > GBP-Part4.tex
 noweave -index -delay Part5.nw > GBP-Part5.tex
 noweave -index -delay Part6.nw > GBP-Part6.tex
}
<CreateTexFromNW 416b>
```

Eine eventuell vorhandene Datei *BuildWithGBP\_html.tex* wird entfernt.

```
416b <CreateTexFromNW1 416b>≡ (416a)

if [-f BuildWithGBP_html.tex]
then
 rm BuildWithGBP_html.tex
fi
}
```

## 50.4. *gitlab-ci.yml* für die Salsa-CI

```

417 <gitlab-ci.yml 417>≡
 variables:
 DEPS: file make texlive noweb texlive-bibtex-extra texlive-lang-german
 texlive-lang-japanese texlive-latex-extra texlive-binaries
 texlive-extra-utils biber tidy texlive-lang-english

 stages:
 - build
 - deploy

 build:
 stage: build
 image: debian:sid
 before_script:
 - apt -y update
 - apt -y install $DEPS
 script:
 - make
 artifacts:
 paths:
 - '*.pdf'
 - '*.epub'
 - 'build-gbp-python-plugin.sh'
 - 'build-gbp-maven-plugin.sh'
 - 'build-gbp-webext-plugin.sh'
 - 'build-gbp.sh'
 - 'build-gbp-java-plugin.sh'
 - 'translation/en_US/target/*.pdf'

 pages:
 stage: deploy
 image: debian:sid
 needs:
 - build
 script:
 - mkdir -p public/de
 - mkdir -p public/en_US
 - cp *.pdf *.epub build-gbp*.sh public/
 - cp *.tex public/de/
 - cp translation/en_US/target/*.pdf public/en_US/

 artifacts:
 paths:
 - public
 only:
 - master

```

**Teil VII.**  
**Anhang**



# Abbildungsverzeichnis

|        |                                                                                          |     |
|--------|------------------------------------------------------------------------------------------|-----|
| 21.1.  | Information zum <code>Java-Team</code> <sup>1</sup> . . . . .                            | 76  |
| 21.2.  | Zugangstoken erstellen <sup>2</sup> . . . . .                                            | 77  |
| 22.1.  | Arbeitsabläufe [38] <sup>3</sup> . . . . .                                               | 80  |
| 31.1.  | Startbildschirm . . . . .                                                                | 107 |
| 31.2.  | Angabe des Projektnamens. . . . .                                                        | 108 |
| 31.3.  | Bye . . . . .                                                                            | 109 |
| 31.4.  | Kein Projektname angegeben. . . . .                                                      | 110 |
| 32.1.  | Keine Konfigurationsdatei gefunden. . . . .                                              | 112 |
| 32.2.  | Name des Quellcode-Paketes . . . . .                                                     | 114 |
| 32.3.  | Namen des Quellpaketes angeben . . . . .                                                 | 114 |
| 32.4.  | Korrekten Namen des Quellpaketes angeben . . . . .                                       | 115 |
| 32.5.  | Ist der Paketname korrekt? . . . . .                                                     | 116 |
| 32.6.  | Ist der Paketname korrekt? . . . . .                                                     | 116 |
| 32.7.  | Korrekten Name des Paketes angeben. . . . .                                              | 117 |
| 32.8.  | Name der Gruppe auf <code>salsa.debian.org</code> angegeben. . . . .                     | 118 |
| 32.9.  | Name der Gruppe auf <code>salsa.debian.org</code> angegeben. . . . .                     | 118 |
| 32.10. | Name der Gruppe auf <code>salsa.debian.org</code> angegeben. . . . .                     | 119 |
| 32.11. | Pfad zum Projektverzeichnis auf der lokalen Maschine . . . . .                           | 120 |
| 32.12. | Pfad zum Projektverzeichnis auf der lokalen Maschine mit <code>OrigName</code> . . . . . | 120 |
| 32.13. | Pfad zum Projektverzeichnis auf der lokalen Maschine (real) . . . . .                    | 121 |
| 32.14. | Soll ein <code>Java</code> -Paket gebaut werden? . . . . .                               | 124 |
| 32.15. | Soll ein <code>Java</code> -Paket mit <code>maven</code> gebaut werden? . . . . .        | 125 |
| 32.16. | Soll eine Erweiterung für Mozilla paketiert werden? . . . . .                            | 126 |
| 32.17. | Soll ein <code>Python3</code> -Paket gebaut werden? . . . . .                            | 127 |
| 32.18. | Gibt es ein übergeordnetes Git-Repository? . . . . .                                     | 132 |
| 32.19. | Ein neues Paket erstellen . . . . .                                                      | 133 |
| 32.20. | Das Git-Repository existiert bereits. . . . .                                            | 135 |
| 32.21. | Name und E-Mail. . . . .                                                                 | 136 |
| 32.22. | Name des Maintainer . . . . .                                                            | 139 |
| 32.23. | E-Mail des Maintainers . . . . .                                                         | 140 |
| 32.24. | <code>Debian-Maintainer</code> OK? . . . . .                                             | 140 |
| 32.25. | Name des <code>Debian-Maintainer</code> . . . . .                                        | 141 |
| 32.26. | E-Mail des <code>Debian-Maintainer</code> . . . . .                                      | 141 |
| 32.27. | E-Mail des <code>Debian-Uploaders</code> . . . . .                                       | 143 |
| 32.28. | E-Mail des <code>Debian-Uploaders</code> . . . . .                                       | 144 |
| 32.29. | Name und E-Mail des Maintainers korrekt? . . . . .                                       | 145 |
| 32.30. | Name des Maintainers eingeben . . . . .                                                  | 146 |
| 32.31. | Name des Maintainers erforderlich . . . . .                                              | 146 |
| 32.32. | E-Mail-Adresse des Maintainers eingeben . . . . .                                        | 147 |

<sup>1</sup>Quelle: <https://salsa.debian.org/java-team/>

<sup>2</sup>Quelle:<https://salsa.debian.org>

<sup>3</sup>©2016 Antoine Beaupré [anarcat@debian.org](mailto:anarcat@debian.org), CC-BY-SA 4.0



|                                                                           |     |
|---------------------------------------------------------------------------|-----|
| 32.33. Dies ist keine E-Mail-Adresse . . . . .                            | 148 |
| 32.34. Ein Git-Repository auf salsa anlegen . . . . .                     | 149 |
| 32.35. Remoteserver ergänzen . . . . .                                    | 150 |
| 32.36. Zeigt Liste der Git-Zweige . . . . .                               | 151 |
| 32.37. Zeigt aktuellen Git-Zweig . . . . .                                | 152 |
| 32.38. Aktueller Git-Zweig . . . . .                                      | 152 |
| 32.39. Vorgegebener Git-Zweig . . . . .                                   | 153 |
| 32.40. Aktueller Git-Zweig . . . . .                                      | 153 |
| 32.41. Auswahl des Debian Release. . . . .                                | 155 |
| 32.42. Name und E-Mail. . . . .                                           | 156 |
| 32.43. Name und E-Mail. . . . .                                           | 157 |
| 32.44. Herunterladen der Debian Sourcen . . . . .                         | 158 |
| 32.45. Eingabe der Url der Debian Sourcen . . . . .                       | 159 |
| 32.46. GPG-Schlüssel verfügbar . . . . .                                  | 161 |
| 32.47. Fingerprint eingeben . . . . .                                     | 162 |
| 32.48. Ist der Fingerprint korrekt? . . . . .                             | 162 |
| 32.49. Korrekten Fingerprint eingeben . . . . .                           | 163 |
| 32.50. Backup des Fingerprints anlegen . . . . .                          | 164 |
|                                                                           |     |
| 33.1. Abfrage - Konfigurationsdatei. . . . .                              | 165 |
| 33.2. Abfrage - Konfigurationsdatei bearbeiten. . . . .                   | 166 |
| 33.3. Programm beendet . . . . .                                          | 166 |
| 33.4. Abfrage - Weiterer Check?. . . . .                                  | 171 |
| 33.5. Auswahl des Debian Zweiges. . . . .                                 | 174 |
| 33.6. Ausgewählter Debian Zweig. . . . .                                  | 175 |
| 33.7. Es gibt nur einen Git-Zweig . . . . .                               | 176 |
| 33.8. Kein Zweig erstellt . . . . .                                       | 177 |
| 33.9. Aufgabenauswahl. . . . .                                            | 178 |
|                                                                           |     |
| 34.1. Herunterladen von salsa.debian.org . . . . .                        | 182 |
| 34.2. Erfolgreich heruntergeladen von salsa.debian.org . . . . .          | 182 |
| 34.3. Kein Herunterladen per uscan von thunderbird.net . . . . .          | 184 |
| 34.4. Kein Herunterladen per uscan von mozilla.org . . . . .              | 185 |
| 34.5. Download - klassisch oder mit uscan . . . . .                       | 185 |
| 34.6. Name des Quellcode-Paketes . . . . .                                | 187 |
| 34.7. Herunterladen (oder kopieren)? . . . . .                            | 188 |
| 34.8. Link für Download eingeben . . . . .                                | 188 |
| 34.9. Download-URL korrekt? . . . . .                                     | 189 |
| 34.10. Korrekte Download-URL . . . . .                                    | 189 |
| 34.11. *.asc Datei herunterladen . . . . .                                | 190 |
| 34.12. Aufforderung zum Kopieren . . . . .                                | 191 |
| 34.13. Kopieren erledigt? . . . . .                                       | 191 |
| 34.14. Programm beenden . . . . .                                         | 192 |
| 34.15. Unbekannte Endung . . . . .                                        | 193 |
| 34.16. Ist dies die korrekte Version? . . . . .                           | 195 |
| 34.17. Welche Version soll gebaut werden? . . . . .                       | 195 |
| 34.18. Wird korrekte Version gebaut? . . . . .                            | 196 |
| 34.19. Programm beenden . . . . .                                         | 196 |
| 34.20. Datei debian/copyright enthält Abschnitt Files-Excluded . . . . .  | 198 |
| 34.21. Dateien ausschließen . . . . .                                     | 198 |
| 34.22. Soll debian/copyright editiert werden? . . . . .                   | 199 |
| 34.23. Name der Datei mit den auszuschließenden Dateien eingeben. . . . . | 200 |

|        |                                                                                                |     |
|--------|------------------------------------------------------------------------------------------------|-----|
| 34.24. | Name der Datei mit den auszuschließenden Dateien. . . . .                                      | 201 |
| 34.25. | Suffix für den Ausschluss von Dateien . . . . .                                                | 202 |
| 34.26. | Eigener Suffix für den Ausschluss von Dateien . . . . .                                        | 202 |
| 34.27. | Warnung! - Kein Suffix angegeben . . . . .                                                     | 203 |
| 34.28. | Zusätzliche Optionen für <i>mk-origtargz</i> . . . . .                                         | 204 |
| 34.29. | <i>debian/copyright</i> enthält auszuschließende Dateien . . . . .                             | 205 |
| 34.30. | Keine Versionsnummer – kein <i>mk-origtargx</i> . . . . .                                      | 206 |
| 34.31. | <i>mk-origtargz</i> gescheitert . . . . .                                                      | 206 |
| 34.32. | Spezielle <i>gbp.conf</i> . . . . .                                                            | 210 |
| 34.33. | Abfrage: Pfad zur <i>gbp.conf</i> . . . . .                                                    | 211 |
| 34.34. | <i>gbp.conf</i> nicht gefunden . . . . .                                                       | 212 |
| 34.35. | Check <i>gbp.conf</i> . . . . .                                                                | 212 |
| 34.36. | Check <i>gbp.conf</i> . . . . .                                                                | 213 |
| 34.37. | <i>gbp.conf</i> zweimal gefunden. . . . .                                                      | 214 |
| 34.38. | Unterschiedliche Konfigurationsdateien . . . . .                                               | 215 |
| 34.39. | <i>gbp.conf</i> im Verzeichnis <i>debian/</i> bearbeiten? . . . . .                            | 215 |
| 34.40. | <i>gbp.conf</i> im Verzeichnis <i>.git/</i> bearbeiten? . . . . .                              | 216 |
| 34.41. | Zweifelhafter Git-Tag . . . . .                                                                | 217 |
| 34.42. | Git Tags entfernen . . . . .                                                                   | 217 |
| 34.43. | Der Zweig ist ... . . . .                                                                      | 219 |
| 34.44. | Importiert wurde der Upstream-Code . . . . .                                                   | 219 |
| 34.45. | Up to date . . . . .                                                                           | 222 |
| 34.46. | Neue Version verfügbar . . . . .                                                               | 222 |
| 34.47. | Kein Repack Suffix in <i>debian/watch</i> . . . . .                                            | 223 |
| 34.48. | Uscan kann Watch-Datei nicht parsen . . . . .                                                  | 224 |
|        |                                                                                                |     |
| 35.1.  | Neue Revision bauen . . . . .                                                                  | 225 |
| 35.2.  | Daten für Maven erstellen? . . . . .                                                           | 228 |
| 35.3.  | Debian-Dateien anzeigen . . . . .                                                              | 229 |
| 35.4.  | Art des Quellpaketes (Nativ? J/N) . . . . .                                                    | 232 |
|        |                                                                                                |     |
| 36.1.  | Es gibt kein Verzeichnis <i>debian/patches</i> . . . . .                                       | 254 |
| 36.2.  | Es gibt ein Verzeichnis <i>debian/patches</i> . . . . .                                        | 254 |
| 36.3.  | Anwendbarkeit der Patches prüfen? . . . . .                                                    | 255 |
| 36.4.  | Patches für Debian erstellen . . . . .                                                         | 256 |
| 36.5.  | Es existiert ein PQ-Zweig. . . . .                                                             | 259 |
| 36.6.  | Alle Patches müssen anwendbar sein. . . . .                                                    | 260 |
| 36.7.  | (Nochmalige) Prüfung der Anwendbarkeit der Patches? . . . . .                                  | 261 |
| 36.8.  | Soll <i>debian/patches/series</i> editiert werden? . . . . .                                   | 262 |
| 36.9.  | Soll <i>gbp pq</i> genutzt werden? . . . . .                                                   | 263 |
| 36.10. | Es gibt Patches . . . . .                                                                      | 264 |
| 36.11. | PQ-Import fehlgeschlagen . . . . .                                                             | 265 |
| 36.12. | Fixed? Retry? . . . . .                                                                        | 266 |
| 36.13. | Kein Import in Patch-Queue . . . . .                                                           | 266 |
| 36.14. | PQ-Import erfolgreich . . . . .                                                                | 267 |
| 36.15. | Der Patch-Queue-Zweig mit Patches von <i>debian/patches</i> wurde ange-<br>wandt.[3] . . . . . | 267 |
| 36.16. | Unterbrechung zum Patchen . . . . .                                                            | 271 |
| 36.17. | Soll <i>gbp pq export</i> angewandt werden? . . . . .                                          | 272 |
| 36.18. | Dateiauswahl bestätigen . . . . .                                                              | 275 |
| 36.19. | <i>dquilt</i> ist nicht konfiguriert. . . . .                                                  | 277 |
| 36.20. | Wird ein Patch benötigt? . . . . .                                                             | 278 |

|                                                                                  |     |
|----------------------------------------------------------------------------------|-----|
| 36.21. Ein weiterer Patch? . . . . .                                             | 278 |
| 36.22. Aufgaben für <code>Quilt</code> . . . . .                                 | 279 |
| 36.23. Name des Patches . . . . .                                                | 280 |
| 36.24. Patch existiert bereits! . . . . .                                        | 281 |
| 36.25. Eine weitere Datei patchen? . . . . .                                     | 283 |
| 36.26. Dateiauswahldialog . . . . .                                              | 284 |
| 36.27. Dateiauswahl bestätigen . . . . .                                         | 285 |
|                                                                                  |     |
| 37.1. Debian-Changelog OK? . . . . .                                             | 292 |
| 37.2. Anzeige der ersten Zeile der Datei <code>debian/changelog</code> . . . . . | 294 |
| 37.3. Aktuelle Version . . . . .                                                 | 294 |
| 37.4. Keine Revisionsnummer angegeben . . . . .                                  | 295 |
| 37.5. Korrekte Versionsbezeichnung angeben? . . . . .                            | 296 |
| 37.6. Korrekte Versionsbezeichnung angeben? . . . . .                            | 297 |
| 37.7. Abfrage des Bezeichners eines nativen Paketes . . . . .                    | 298 |
| 37.8. Abfrage des Bezeichners . . . . .                                          | 299 |
| 37.9. Weitere Optionen für <code>dch</code> . . . . .                            | 300 |
| 37.10. Da läuft was falsch! . . . . .                                            | 303 |
| 37.11. Auswahl des Branches . . . . .                                            | 304 |
| 37.12. Anzeige des <code>Git</code> -Zweiges . . . . .                           | 305 |
| 37.13. Release-Branch der Distribution . . . . .                                 | 306 |
| 37.14. Auswahl des Build-Systems . . . . .                                       | 308 |
| 37.15. Parameterüberprüfung . . . . .                                            | 309 |
| 37.16. Bau des Paketes starten . . . . .                                         | 309 |
| 37.17. Beenden . . . . .                                                         | 310 |
| 37.18. Information zur Aktualisierung der Build-Umgebung . . . . .               | 310 |
| 37.19. <code>.sbuilddrc</code> prüfen . . . . .                                  | 312 |
| 37.20. Auswahl der zu erstellenden Cow. . . . .                                  | 314 |
| 37.21. Anzeige der Version mit Revisionsnummer . . . . .                         | 317 |
| 37.22. Anzeige der Revisionsnummer . . . . .                                     | 318 |
| 37.23. Soll der Upstream-Tarball auch hochgeladen werden . . . . .               | 318 |
| 37.24. Anzeige der Optionen von <code>gbp buildbackage</code> . . . . .          | 319 |
| 37.25. Erfolgreicher Bauversuch! . . . . .                                       | 321 |
|                                                                                  |     |
| 40.1. Upload des Release vorbereiten . . . . .                                   | 328 |
| 40.2. <code>*.changes</code> -Datei auswählen . . . . .                          | 330 |
| 40.3. Lintian: All Well? . . . . .                                               | 332 |
| 40.4. Paket ist aktuell . . . . .                                                | 333 |
| 40.5. Älteres Paket verfügbar . . . . .                                          | 334 |
| 40.6. Uscan - OK? . . . . .                                                      | 334 |
| 40.7. Uscan schlägt fehl . . . . .                                               | 335 |
| 40.8. Unterschiede feststellen . . . . .                                         | 336 |
| 40.9. Zu viele Versionen ausgewählt . . . . .                                    | 337 |
| 40.10. Zu wenig ausgewählt . . . . .                                             | 338 |
| 40.11. Umkehr der Reihenfolge der zu vergleichenden Pakete . . . . .             | 339 |
|                                                                                  |     |
| 41.1. Kein Changelog - kein Hochladen . . . . .                                  | 344 |
| 41.2. Changelog veröffentlichungsreif? . . . . .                                 | 345 |
| 41.3. Branch überprüfen . . . . .                                                | 346 |
| 41.4. Name der Distribution eingeben . . . . .                                   | 347 |
| 41.5. Name der Distribution überprüfen . . . . .                                 | 347 |
| 41.6. Fürs Release bauen . . . . .                                               | 348 |

|        |                                                                   |     |
|--------|-------------------------------------------------------------------|-----|
| 41.7.  | Fürs Release bauen . . . . .                                      | 349 |
| 41.8.  | DebDiff benötigt? . . . . .                                       | 350 |
| 42.1.  | Fürs Release bauen . . . . .                                      | 352 |
| 42.2.  | Patch-Queue Zweige gefunden . . . . .                             | 353 |
| 42.3.  | Letzter Halt - Alles OK? . . . . .                                | 354 |
| 43.1.  | Ziel des Uploads. . . . .                                         | 358 |
| 43.2.  | Ist das angegebene Ziel des Uploads korrekt? . . . . .            | 360 |
| 43.3.  | Bye . . . . .                                                     | 360 |
| 43.4.  | Signieren erfolgreich? . . . . .                                  | 361 |
| 43.5.  | Upload to FTP-Master - OK? . . . . .                              | 363 |
| 43.6.  | Soll ein Binär-Paket auf FTP-Master hochgeladen werden? . . . . . | 364 |
| 43.7.  | Wirklich nach Experimental hochladen? . . . . .                   | 365 |
| 43.8.  | Soll das Hochladen auf FTP-Master simuliert werden? . . . . .     | 366 |
| 43.9.  | Ist alles ok mit dem Hochladen? . . . . .                         | 366 |
| 43.10. | Tage der Verzögerung . . . . .                                    | 368 |
| 47.1.  | Maven Plugin geladen . . . . .                                    | 386 |
| 47.2.  | Pfad zur Maven-Chroot ermitteln . . . . .                         | 387 |
| 47.3.  | Pfad zur Maven-Chroot angeben . . . . .                           | 388 |
| 47.4.  | Arbeitsverzeichnis in der Maven-Chroot ermitteln . . . . .        | 389 |
| 47.5.  | Arbeitsverzeichnis in der Maven-Chroot angeben . . . . .          | 390 |
| 47.6.  | Maven-Chroot existiert nicht . . . . .                            | 391 |
| 48.1.  | Webext-Plugin wurde geladen . . . . .                             | 404 |
| 49.1.  | Python-Plugin geladen . . . . .                                   | 406 |



# Literaturverzeichnis

- [1] Creative Commons. „Attribution-ShareAlike 4.0 International“. In: <https://creativecommons.org/licenses/by-sa/4.0/legalcode>. (15. Okt. 2013). URL: <https://creativecommons.org/licenses/by-sa/4.0/legalcode>. (Siehe S. 3).
- [2] Free Software Foundation, Inc. „GNU General Public License 3“. In: <https://www.gnu.org/licenses/gpl-3.0.de.html>. (29. Juni 2007). URL: <https://www.gnu.org/licenses/gpl-3.0.de.html>. (Siehe S. 3).
- [3] Guido Günther. „Building Debian Packages with git-buildpackage“. In: *Git-Buildpackage* (2017). URL: <https://honk.sigxcpu.org/projects/git-buildpackage/manual-html/> (siehe S. 7, 22, 33, 72, 267, 272).
- [4] Norman Ramsey. „Noweb — A Simple, Extensible Tool for Literate Programming“. In: *NoWeb* (28. Juni 2018). URL: <https://www.cs.tufts.edu/~5Ctextasciitilde%20nr/noweb/> (siehe S. 10).
- [5] *Simple Packaging Tutorial*. 26. Okt. 2019. URL: <https://wiki.debian.org/SimplePackagingTutorial> (siehe S. 21, 23).
- [6] Debian Project. „Die Debian-Richtlinien für Freie Software (DFSG)“. German. In: *Debian-Gesellschaftsvertrag* (26. Apr. 2004). URL: [https://www.debian.org/social\\_contract.de.html](https://www.debian.org/social_contract.de.html) (siehe S. 21, 27, 30, 31).
- [7] Ian Jackson, Christian Schwarz und David A. Morris. „Debian Policy“. English. Version 4.6.1. In: *Debian Policy Manual* (22. Mai 2022). Hrsg. von Die Debian-Policy-Gruppe. URL: <https://www.debian.org/doc/debian-policy/>. GNU General Public License Version 2+ (siehe S. 21, 23, 27, 29–31, 51, 57, 231, 235, 244, 250, 251, 380).
- [8] Christopher Yeoh, Paul 'Rusty' Russell, Daniel Quinlan. „Filesystem Hierarchy Standard“. English. Version 3.0. In: *Filesystem Hierarchy Standard* (19. März 2015). Hrsg. von The Linux Foundation LSB Workgroup. URL: <https://www.debian.org/doc/packaging-manuals/fhs/fhs-3.0.html>. BSD (siehe S. 21).
- [9] Ian Jackson u. a. „Debian-Entwicklerreferenz“. Deutsch. Version 13.3. In: *Debian Entwicklerreferenz* (5. Aug. 2023). Hrsg. von Hideki Nussbaum Lucas and Yamane und Holger Levsen. URL: <https://www.debian.org/doc/manuals/developers-reference/index.de.html>. GNU General Public License Version 2+ (siehe S. 21, 79, 80, 82, 94).
- [10] Osamu Aoki. „Leitfaden für Debian-Betreuer. debmake-doc“. Deutsch. In: *Debmake-Doc* (26. März 2019). URL: <https://www.debian.org/doc/devel-manuals#debmake-doc>. Expat-Lizenz (siehe S. 22).
- [11] Josip Rodin, Osamu Aoki. „Debian-Leitfaden für Neue Paketbetreuer. New Maintainer Guide“. Deutsch. Version 1.2.43. In: *Debian-Leitfaden für Neue Paketbetreuer* (7. Nov. 2020). Hrsg. von Osamu Aoki. wird ersetzt durch Aoki, Leitfaden für Debian-Betreuer. URL: <https://www.debian.org/doc/manuals/maint-guide/>. GNU General Public License Version 2+ (siehe S. 22, 35, 60, 231, 293).
- [12] Axel Beckert und Frank Hofmann. „Debian-Paketmanagement“. In: *DPMB* (7. Feb. 2021), S. 400. URL: <https://book.dpmb.org/debian-paketmanagement.chunked/index.html> (siehe S. 22, 24, 25).



- [13] wiki.debian.org, Hrsg. *What is a "package"?* 23. Dez. 2020. URL: <https://wiki.debian.org/Packaging/Intro> (siehe S. 23).
- [14] *ReproducibleBuilds*. 6. Dez. 2020. URL: <https://reproducible-builds.org/> (siehe S. 23).
- [15] *The experimental repository*. 15. Nov. 2020. URL: <https://wiki.debian.org/DebianExperimental> (siehe S. 24).
- [16] *The Debian GNU/Linux FAQ*. 12. Aug. 2019. URL: <https://www.debian.org/doc/manuals/debian-faq/index.de.html> (siehe S. 24, 35).
- [17] *How can i help*. 7. Feb. 2021. URL: <https://wiki.debian.org/how-can-i-help> (siehe S. 25).
- [18] *Teams*. 10. Jan. 2024 (siehe S. 25, 138).
- [19] Steve Langasek. „DEP-5: Machine-readable debian/copyright“. In: *Machine-readable debian/copyright* (24. Feb. 2012). URL: <https://dep-team.pages.debian.net/deps/dep5/> (siehe S. 27, 28, 31, 199, 234).
- [20] *TeamsFTPMaster*. 13. März 2020. URL: <https://wiki.debian.org/TeamsFTPMaster> (siehe S. 27).
- [21] *CopyrightReviewTools*. 17. Dez. 2021. URL: <https://wiki.debian.org/CopyrightReviewTools> (siehe S. 27).
- [22] Mathew Palmer. „Debian-Mentors FAQ“. English. In: *Debian-Wiki* (2007). URL: <https://wiki.debian.org/DebianMentorsFaq>. GNU General Public License version 2 (siehe S. 29, 32).
- [23] Jilayne et al. Lovejoy. „Open Source License Compliance Handbook“. In: *Open Source License* (29. Apr. 2019). URL: <https://github.com/finos/OSLC-handbook/tree/master/output> (siehe S. 29).
- [24] *Using Quilt*. 22. Juli 2020. URL: <https://wiki.debian.org/UsingQuilt> (siehe S. 32).
- [25] Andreas Grünbacher. „How to Survive With Many Patches or Introduction to Quilt“. In: *Introduction to Quilt* (22. Feb. 2012). URL: <http://users.suse.com/~agruen/quilt.pdf> (siehe S. 32).
- [26] Raphael Hertzog. „DEP-3: Patch Tagging Guidelines“. In: *Patch Tagging Guidelines* (26. Nov. 2009). URL: <https://dep-team.pages.debian.net/deps/dep3/> (siehe S. 33, 271).
- [27] *Git-Mailinfo -Manpage*. 20. Apr. 2020. URL: <https://manpages.debian.org/unstable/git-man/git-mailinfo.1.en.html> (siehe S. 33).
- [28] Niels Thykier u. a. „Debian Policy for Java“. In: *Debian Java Policy* (27. Juli 2020). URL: <https://www.debian.org/doc/packaging-manuals/java-policy/> (siehe S. 39, 40, 97, 248).
- [29] Markus Koschany. „Packaging Java with Javatools“. In: <https://people.debian.org/~apo/java/tutorial.html> (2. Aug. 2018). URL: <https://people.debian.org/%5Ctextasciitilde%7B%7Dapo/java/tutorial.html> (siehe S. 39).
- [30] Torsten Werner [twerner@debian.org](mailto:twerner@debian.org) Niels Thykier [niels@thykier.net](mailto:niels@thykier.net) Javier Fernández-Sanguino Peña [jfs@debian.org](mailto:jfs@debian.org) Sylvestre Ledru [sylvestre@debian.org](mailto:sylvestre@debian.org). „Debian Java FAQ.“ In: *Debian Java FAQ*. (22. Mai 2014). URL: <https://www.debian.org/doc/manuals/debian-java-faq/> (siehe S. 39).
- [31] *Help the Java Team distribute your project*. 31. Jan. 2019. URL: <https://java.debian.net/blog/2019/01/help-the-java-team-distribute-your-project.html> (siehe S. 39).

- [32] *Introduction to the Standard Directory Layout*. 29. Dez. 2020. URL: <https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html> (siehe S. 41).
- [33] *Devsripts*. 28. Nov. 2020. URL: <https://manpages.debian.org/unstable/devscripts/devscripts.1.en.html> (siehe S. 57).
- [34] *SBuild*. 15. Mai 2024. URL: <https://wiki.debian.org/sbuild> (siehe S. 68, 310).
- [35] *debian mit debootstrap in chroot-Umgebung installieren*. 16. Feb. 2014. URL: [http://www.kai-hildebrandt.de/linux/debian%5C\\_chroot.html](http://www.kai-hildebrandt.de/linux/debian%5C_chroot.html) (siehe S. 69).
- [36] *Salsa*. 22. Feb. 2023. URL: <https://wiki.debian.org/Salsa> (siehe S. 75).
- [37] *Salsa-Doc*. 26. Aug. 2023. URL: [wiki.debian.org/Salsa/Doc](https://wiki.debian.org/Salsa/Doc) (siehe S. 75).
- [38] [wiki.debian.org](https://wiki.debian.org). „Debian Releases“. In: <https://wiki.debian.org/DebianReleases>. 29. Nov. 2020. URL: <https://salsa.debian.org/debian/package-cycle/raw/master/package-cycle.svg> (siehe S. 80).
- [39] *ReleaseTeam*. 22. März 2021. URL: <https://wiki.debian.org/Teams/releaseTeams> (siehe S. 93).
- [40] Debian, Hrsg. *Lintian User's Manual*. 23. Jan. 2023. URL: <https://lintian.debian.org/manual/index.html> (siehe S. 97).
- [41] *GBP-Import-orig uscan*. 19. Juli 2022. URL: <https://manpages.debian.org/unstable/git-buildpackage/gbp-import-orig.1.en.html> (siehe S. 224).
- [42] [wiki.debian.org](https://wiki.debian.org). „Upstream METadata GATHERed with YAml (UMEGAYA)“. In: <https://wiki.debian.org/UpstreamMetadata?action=recall&rev=138>. 31. Jan. 2020. URL: <https://wiki.debian.org/UpstreamMetadata?action=recall&rev=138> (siehe S. 233).
- [43] Charles Plessy und Andreas Tille. „DEP-12: Per-package machine-readable metadata about Upstream“. In: *Per-package machine-readable metadata about Upstream* (23. Feb. 2014). URL: <https://dep-team.pages.debian.net/deps/dep12/> (siehe S. 233).
- [44] Machine-Readable Copyright. „Machine-readable debian/copyright file“. In: *Debian Policy* (17. Nov. 2020). URL: <https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/%5C#stand-alone-license-paragraph> (siehe S. 235).
- [45] Debian Projekt, Hrsg. *Manpage uscan*. 4. Aug. 2019. URL: <https://manpages.debian.org/buster/devscripts/uscan.1.en.html> (siehe S. 239).
- [46] *GBP-PQ*. 1. Juli 2020. URL: <https://manpages.debian.org/unstable/git-buildpackage/gbp-pq.1.en.html> (siehe S. 268).





# Stichwortverzeichnis

- LaTeX, 10
- TeX-Dateien, 416
- ~/.bashrc, 60, 70
  
- Abhängigkeit, 29, 39
- Aktualisierung, 47
- ant, 45
- apt, 36
  
- Build-Umgebungen, 32
  
- Chroot, 60
- Chroot (maven), 42, 69
- cme, 28
- compare-version, 35
- Copyright-Review, 27
- cowbuilder, 60
- cowdancer, 60
- Creative Commons, 3
  
- DEBEMAIL, 60, 136
- DEBFULLNAME, 60, 136
- debhelper, 22, 236
- Debian-Entwicklerreferenz, 21
- Debian-Keyring, 161
- Debian-Paket, 23
- Debian-Policy, 21, 27, 30
- Debian-Projekt, 23
- debian/watch, 221, 239
- debmake, 27
- devscripts, 36, 57
- DFSG, 27, 30
- dh\_make, 37, 57
- Distribution, 9
- dpkg, 36
- dquilt, 69
- dversionmangle, 241
  
- E-Book, 10
- Entwicklerreferenz, 21
- EPUB-Dokument, 10
- experimental, 24
  
- Fehlerbehebung, 32
  
- FHS, 21
- filenamemangle, 242
- Fingerprint, 60, 161
- Freie Software, 27
- FTBFS, 60
  
- gbp buildpackage, 60
- gbp pq, 33
- gbp.conf, 71, 72, 221
- Geany, 10
- Gesellschaftsvertrag, 21
- git, 10
- git-buildpackage, 10, 22, 57
- Git-Repositorium, 135
- GNU General Public License, 3
- GPG-Schlüssel, 57, 219, 223
  
- Handbuch, 22
- Hauptprogramm, 106
- Hilfsprogramme, 37
- Hochladen, 343
- Homeverzeichnis, 111
  
- Java-Anwendung, 39
- Java-Bibliothek, 39, 40
- Java-Build-System, 41
- Java-FAQ, 39
- Java-Policy, 39
- Java-Programm, 40
- javahelper, 39
  
- Konfigurationsdatei, 59, 105, 110, 161, 165
- Kurzanleitung, 15
  
- Leitfaden f. neue Betreuer, 22
- Literate Programming, 10
- Literatur, 22
- Lizenz, 3, 27
- Lizenzen, 27, 29
- Lizenzprüfung, 27, 335
- Log-Datei, 129
  
- Mail-Extension, 48

#### 4. Mai 2025

main, 27  
Maintainer, 138  
Maintainer-Schlüssel, 60  
Maven, 41, 42  
maven, 45  
mh\_make, 42  
  
noweave, 416  
noweb, 10  
  
oldoldstable, 24  
oldstable, 24  
Optionen (uscan), 240  
Original-Autor, 28  
Originalquelle, 36  
  
Paketieren, 7  
Paketmanagementsystem, 47  
Paketverwaltungswerkzeug, 235  
Patchen, 32  
pbuilder, 60  
PDF-Dokument, 10  
Perl-Format, 239  
pom.xml, 41  
pristine-tar, 156  
Programmablauf, 105, 106  
Programmskript, 105  
Projekt, 107  
Projektname, 107  
projektspezifisch, 59  
Proposed-Updates, 80, 81  
  
Quelltext, 10  
quilt, 33, 69  
  
reguläre Ausdrücke, 239  
Release-Team, 81  
Reproduzierbarkeit, 23  
Revisionsnummer, 35  
Rückportierung, 80  
  
Security-Patches, 32, 80  
security-Team, 80  
signieren, 60  
stable, 23  
Standard-Version, 236  
Startdialog, 105  
Systemnutzer, 47  
  
testing, 23  
Thunderbird, 48  
  
unstable, 23  
Uploader, 138  
  
uscan, 36, 221, 239  
uversionmangle, 241  
  
Vergleichsregeln, 35  
Versionierungsindex, 241  
Versionierungsschema, 35, 194  
Versionsbezeichnung, 35, 201  
Verzeichnis, 129  
Verzeichnisstruktur Maven, 41  
Veröffentlichung, 10, 179  
  
Watch (Datei), 36, 239  
Werkzeuge, 10  
  
Zugangstoken, 76, 409