

---

# Debian Repository mit debarchiver HOWTO

Ein eigenes und komfortables APT(-GET-bares)-Repository mit debarchiver aufsetzen.

Daniel Leidert <daniel.leidert@wgdd.de>

Copyright © 2004, 2005, 2006 Daniel Leidert

2005-12-03

## Versionsgeschichte

Version 0.96                      2005-12-03                      dale

Inhaltliche Umstrukturierungen. Variablen sind direkt und unter dem eigenen Namen verlinkt. Das sollte das Nachschlagen einfacher machen. Einige denglische Begrifflichkeiten wurden ersetzt.

Version 0.95                      2005-10-20                      dale

Informationen zum Erstellen eines signierten Repository wurden hinzugefügt und die entsprechenden Punkte aus der ToDo-Liste gestrichen. Diverse Typos konnten beseitigt und Informationen aktualisiert werden.

Version 0.94                      2005-05-25                      dale

Neue Optionen aufgenommen (Paket-Version 0.3.1) und Beschreibungen/Tipps zum Einsortieren von Paketen in spezielle Distributionen aufgenommen.

Version 0.93                      2005-04-30                      dale

Weitere Arbeit an der Apache Konfiguration, sowie dem Zugriff auf das Repository.

Version 0.92                      2005-04-30                      dale

Weitere Typos beseitigt, einige Erklärungen zur Funktionsweise hinzugefügt, strukturelle Veränderungen zur besseren Gliederung, Abarbeitung einiger Punkte der TODO-Liste (speziell die Integration von GnuPG).

Version 0.1                        2004-11-15                      dale

Initialer Import.

## Zusammenfassung

\$Id: howto-aptrep.de.xml,v 1.25 2006/01/11 01:23:30 dleiert Exp \$

Da, wo das Debian Repository *HOWTO* aufhört, fangen viele Fragen erst an. Sehr häufig findet man die Frage nach der Erstellung eines eigenen Repository. Und wie schön wäre es, wenn bereits fertige Lösungen existieren würden? Sie existieren. Daher versucht dieses *HOWTO* an genau dieser Stelle anzusetzen.

Ziel dieses *HOWTO*s ist es, den Aufbau eines APT-Repositorys und anschließend die Einrichtung eines lokalen oder entfernten Repositorys mittels debarchiver<sup>[1]</sup> zu erklären. Das Repository soll dabei einfach und komfortabel verwaltet werden können.

---

<sup>[1]</sup><http://packages.debian.org/debarchiver>

# Inhaltsverzeichnis

1. Vorwort .....	4
1.1. Copyright und Lizenz .....	4
1.2. Haftungsausschluss .....	4
1.3. Todo .....	5
1.4. Credits und Danksagung .....	5
1.5. Feedback .....	5
2. Einleitung .....	6
2.1. Der Aufbau eines APT-Repositoriums .....	6
2.2. Warum debarchiver? .....	6
2.2.1. Alternativen .....	7
3. debarchiver installieren und einrichten .....	8
3.1. Struktur und Idee des Programmes .....	8
3.2. debarchiver konfigurieren .....	8
3.3. Funktionsweise und Hinweise zur Nutzung .....	11
3.3.1. debarchiver und apt-ftpparchive .....	11
3.3.2. debarchiver und dpkg-scanpackages/dpkg-scansources .....	12
3.3.3. Sortierung der Pakete .....	12
3.4. [Optional] Signaturen mit GnuPG verifizieren .....	13
3.5. [Optional] Repository signieren .....	14
4. Arbeiten mit dem Repository .....	16
4.1. Pakete zum Repository hinzufügen .....	16
4.1.1. Ein lokales Repository bedienen .....	16
4.1.2. Ein entferntes Repository bedienen .....	17
4.2. Pakete aus dem Repository entfernen .....	18
4.3. Pakete apt-get(ten) .....	18
4.4. Integrität des Repository verifizieren .....	19
4.5. [Optional] Pakete über http:// verfügbar machen .....	19
4.6. [Optional] Eine Übersicht aller vorhandenen Pakete (on-the-fly) erstellen .....	20
4.7. [Optional] Mehr ... .....	21

# 1. Vorwort

Ziel dieses *HOWTO*s ist es, den Aufbau eines (non-pooled) APT-Repositoriums und anschließend die Einrichtung eines lokalen oder entfernten Repository für eigene Pakete mittels debarchiver <sup>[1]</sup> zu erklären. Das Repository soll automatisiert arbeiten und somit einfach und komfortabel verwaltet werden können. Weitere Informationen und diverse *HOWTO*s zu debarchiver findet man auf der der Projekt-Homepage <sup>[2]</sup>.

Es ist nicht das Ziel dieses *HOWTO*s, die folgenden Fragen zu (er)klären:

- wie Debian-Pakete erstellt werden. Für dieses Problem empfehle ich einen Blick in den *Debian New Maintainers' Guide* <sup>[3]</sup> (mehrsprachig verfügbar, Englisch meist am aktuellsten).
- wie ein offizielles Debian Repository gespiegelt wird. Für dieses Problem empfehle ich einen Blick in das Paket debmirror <sup>[4]</sup> oder reprepro <sup>[5]</sup>.
- offizielle Pakete anderen PCs (z.B. Systeme mit langsamer Internet-Anbindung in einem Netzwerk) zur Verfügung zu stellen. Für dieses Problem empfehle ich einen Blick in das Paket apt-proxy <sup>[6]</sup> oder auch apt-move <sup>[7]</sup>.

## 1.1. Copyright und Lizenz

This document, »APT-Repository mit debarchiver *HOWTO*«, is copyrighted (C) 2004-2006 by Daniel Leidert. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License <sup>[8]</sup>, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts.

All copyrights are held by their respective owners, unless specifically noted otherwise. Use of a term in this document should not be regarded as affecting the validity of any trademark or service mark. Naming of particular products or brands should not be seen as endorsements.

## 1.2. Haftungsausschluss

Der Autor übernimmt keinerlei Gewähr für die Aktualität, Korrektheit, Vollständigkeit und/oder Qualität der bereitgestellten Informationen. Haftungsansprüche gegen den Autor, welche sich auf Schäden materieller oder ideeller Art beziehen, die durch die Nutzung oder Nichtnutzung der dargebotenen Informationen bzw. durch die Nutzung fehlerhafter und/oder unvollständiger Informationen verursacht wurden, sind grundsätzlich ausgeschlossen.

---

<sup>[2]</sup><http://www.opal.dhs.org/programs/debarchiver/index.oml>

<sup>[3]</sup><http://www.debian.org/doc/maint-guide/>

<sup>[4]</sup><http://packages.debian.org/debmirror>

<sup>[5]</sup><http://packages.debian.org/reprepro>

<sup>[6]</sup><http://packages.debian.org/apt-proxy>

<sup>[7]</sup><http://packages.debian.org/apt-move>

<sup>[8]</sup><http://www.gnu.org/copyleft/fdl.html>

## 1.3. Todo

Diese Dinge müssen noch hinzugefügt werden:

- mehrere Archive verwalten <sup>[9]</sup>
- aktuell: .changes-Dateien signieren (signed... key) mit `debsign -m"xyz" package.changes`

## 1.4. Credits und Danksagung

Ein großer Dank geht an Ola Lundqvist für dieses sehr nützliche Tool und diverse hilfreiche Diskussionen. Dank gebührt ebenfalls Jarno Elonen für sein `parse-apt-files.inc` Skript <sup>[10]</sup> und seine Hilfestellungen bei diversen Fixes.

## 1.5. Feedback

Feedback ist gerne willkommen. Zusätze, Verbesserungen, Anregungen oder Kritik kannst du bei Bedarf an [<daniel.leidert@wgdd.de>](mailto:daniel.leidert@wgdd.de) senden. Quellcode-Patches, Wünsche oder Verbesserungen, die das Programm selbst betreffen, können an [Ola Lundqvist](#) gesendet werden.

---

<sup>[9]</sup> [/usr/share/doc/debarchiver/README](#)

<sup>[10]</sup> <http://elonen.iki.fi/code/parse-apt-files.inc>

## 2. Einleitung

Um das HOWTO besser nutzen und verstehen zu können, ist es notwendig, an dieser Stelle zuerst einige Konventionen zu erklären.



### Wichtige Konventionen

```
$ Befehl
```

Dieser Befehl kann als Benutzer ausgeführt werden. Es sollten keine speziellen Berechtigungen notwendig sein.

```
# Befehl
```

Dieser Befehl benötigt Root-Rechte(!). Solche Befehle sollten niemals stur abgetippt oder bedenkenlos mittels Copy&Paste auf die Konsole übertragen werden.

```
// Kommentar
$ Befehl // Kommentar
```

Kommentar zu einem Schritt oder eine Anweisung, die sich nicht direkt auf die Kommandozeile bezieht.

## 2.1. Der Aufbau eines APT-Repositoriums

Der Aufbau eines typischen Debian Repository ist im *Debian Repository HOWTO* <sup>[12]</sup> sehr ausführlich erklärt. Wer des Englischen nicht mächtig ist, kann auf die deutsche Übersetzung <sup>[13]</sup> von Patrick Schönfeld zurückgreifen. Als Erweiterung empfiehlt sich das *Automatic Debian Package Repository HOWTO* <sup>[14]</sup> von Robert Sanchez.

Es wird vorausgesetzt, dass das *Debian Repository HOWTO* gelesen wurde, da im folgenden nicht auf alle Termini eingegangen wird, die bei Kenntnis des *HOWTO* bekannt sein sollten. Falls das *HOWTO* noch nicht gelesen wurde, sollte dies vor dem weiteren Lesen getan werden.

## 2.2. Warum debarchiver?

Debarchiver <sup>[1]</sup> ist ein Tool, das Debian-Pakete (mit und ohne Quellen), die in ein spezielles Verzeichnis (`incoming/` ) kopiert werden, automatisch sortiert. Dadurch lässt es sich sowohl lokal, als auch auf einem entfernten System benutzen. Gleichzeitig generiert es mittels `dpkg-scanpackages/dpkg-scansources` <sup>[15]</sup> oder `apt-ftparchive` <sup>[16]</sup> die notwendigen Dateien (`Sources.gz Packages.gz Release`), um ein arbeitsfähiges APT-Repository zu erhalten. Über eine Konfigurationsdatei lässt sich zudem festlegen, welchen Inhalt die `Release` Dateien und welche Struktur das Repository bekommen soll. `debarchiver` verwaltet quasi das Archiv fast allein und eignet sich for allem für mittlere Archiv-Größen. Folgende Funktionen sind bis einschließlich Paket-Version 0.5.3 (noch) nicht oder nur eingeschränkt vorhanden:

- automatisches Löschen oder Verschieben bestimmter Dateien oder Pakete ([Abschnitt 4.2, „Pakete aus dem Repository entfernen“](#))
- eine Pool-Struktur anlegen (ist aber angedacht)

<sup>[12]</sup><http://www.debian.org/doc/manuals/repository-howto/repository-howto.html>

<sup>[13]</sup><http://www.patrick-urlaub.de/repository-howto.de.html>

<sup>[14]</sup><http://familiasanchez.net/~sanchezr/?page=debrepository>

<sup>[15]</sup><http://packages.debian.org/dpkg-dev>

<sup>[16]</sup><http://packages.debian.org/apt-utils>

## 2.2.1. Alternativen

Neben debarchiver <sup>[1]</sup> existieren verschiedene andere Programme <sup>[17]</sup>, um ein Repository zu erstellen. Eine kleine Übersicht:

- mini-dinstall <sup>[18]</sup> <sup>[19]</sup> <sup>[20]</sup>

(vor allem für kleine Repositories geeignet; strukturell sehr einfach gehalten, da alle Dateien in einem Verzeichnis liegen; Fernadministration über »Incoming«-Mechanismus möglich)

- debpool <sup>[21]</sup>

(Pool-Struktur entsprechend Debian; Binary-Pakete können nur per Hand hinzugefügt werden; signierte Release-Dateien möglich; signierte Pakete können auf die Signatur geprüft werden; Fernadministration über »Incoming«-Mechanismus möglich)

- reprepro <sup>[5]</sup> alias mirroring <sup>[22]</sup>

(Fernadministration evtl. möglich, aber besser geeignet für lokale Repositories; arbeitet mit einem Datenbank-Backend; kann Debian-Repositories vollständig oder teilweise spiegeln; Pool-Struktur entsprechend Debian; kann Pakete zwischen den Distributionen verschieben; sehr vielfältige Funktionalität)

- dak-Skripte <sup>[23]</sup> <sup>[24]</sup> (Katie, ...)

(enthält die Skripte, die vom Debian-Projekt genutzt werden; lohnt sich weder für kleine, noch für mittlere Repositories; Pool-Struktur entsprechend Debian; Funktionalität äußerst umfangreich; kann Pakete zwischen den Distributionen verschieben)

---

<sup>[17]</sup><http://wiki.debian.org/HowToSetupADebianRepository>

<sup>[18]</sup><http://packages.debian.org/mini-dinstall>

<sup>[19]</sup><http://people.debian.org/~calvin/unofficial/>

<sup>[20]</sup><http://usr/share/doc/dput/FAQ>

<sup>[21]</sup><http://packages.debian.org/debpool>

<sup>[22]</sup><http://mirroring.alioth.debian.org/>

<sup>[23]</sup><http://ukai.org/wiliki/wiliki.cgi?Debian%3Adak&l=jp>

<sup>[24]</sup><http://packages.debian.org/dak>

---

## 3. debarchiver installieren und einrichten

debarchiver wird einfach mittels **apt-get** installiert:

```
# apt-get install debarchiver
```

Das Programm ist nun installiert. Im nächsten Schritt, sollte man sich mit [Abschnitt 3.1, „Struktur und Idee des Programmes“](#) und der [Konfigurationsdatei debarchiver.conf](#) vertraut machen.

### 3.1. Struktur und Idee des Programmes

Betrachten wir zunächst die Struktur des Verzeichnisses `/var/lib/debarchiver` :

```
# ls -lA /var/lib/debarchiver
drwxr-xr-x 4 debarchiver debarchiver 128 2004-11-17 15:25 dists
drwxrwsr-x 5 root          debarchiver 120 2004-11-17 15:25 incoming
```

Das Verzeichnis `dists` enthält bzw. ist das eigentliche Repository. Im Verzeichnis `incoming` werden alle neuen Pakete (mit Quellen bzw. `.changes` Datei) abgelegt und entsprechend ihrer Zugehörigkeit zu einer Distribution (Feld: »Distribution:«) einsortiert. In bestimmten Zeitabständen, die man selbst festlegen kann, schaut **debarchiver** in diesem Verzeichnis nach neuen Paketen und sortiert diese dann ein.

Unterhalb von `incoming` befinden sich standardmäßig 3 weitere Verzeichnisse (siehe auch [%distinputdirs](#)). In älteren debarchiver-Paketen (< 0.1.8), sind diese 3 Verzeichnisse nur für Root schreibbar, was verschiedentlich zu Problemen führt. In diesem Fall gibt man der Gruppe »debarchiver« zusätzlich Schreibrechte:

```
# ls -lA /var/lib/debarchiver/incoming
drwxrwxr-x 2 root debarchiver 48 2004-10-18 22:30 stable
drwxrwxr-x 2 root debarchiver 48 2004-10-18 22:30 testing
drwxrwxr-x 2 root debarchiver 48 2004-11-15 04:05 unstable
// der Gruppe »debarchiver« Schreibrechte geben, falls nicht vorhanden (optional)
# chmod g+w /var/lib/debarchiver/incoming/*
```

In diesen Verzeichnissen können Pakete abgelegt werden, für die:

Damit man auch als Benutzer Pakete in diesen Verzeichnissen ablegen kann, wird der Benutzer der Gruppe »debarchiver« hinzugefügt:

```
# adduser user debarchiver
```

Alternativ kann ein Benutzer seine eigene [Konfigurationsdatei debarchiver.conf](#) anlegen und Pfade auswählen, in denen er Schreibrechte besitzt. Es ist also möglich, mehrere Repositories mit **debarchiver** zu erstellen. Hierzu gibt die Dokumentation <sup>[9]</sup> mehr Auskunft.

### 3.2. debarchiver konfigurieren

Der Einfachheit halber gehen wir in den nächsten Schritten davon aus, dass als *Incoming-Verzeichnis* `/var/lib/debarchiver/incoming` verwendet werden soll.

Die Konfigurationsdatei `/etc/debarchiver.conf` ist an sich gut dokumentiert. Zusätzlich habe ich eine [Beispielconfig](#) mit deutschen Kommentaren versehen. Diese Datei kann verwendet werden. Sie erstellt jedoch ein umfangreiches Repository für oldstable (Woody), stable (Sarge), testing (Etch) und unstable (Sid). Die Datei hat folgenden Inhalt:

<sup>[25]</sup> <http://www.stanchina.net/~flavio/fglrx-installer.html>

<sup>[26]</sup> <http://www.opera.com>

## Beispiel 1. Konfigurationsdatei für debarchiver

```
# debarchiver.conf

$destdir = "/var/lib/debarchiver/dists";
$inputdir = "/var/lib/debarchiver/incoming";
$cachedir = "/var/cache/debarchiver";
$copycmd = "cp -af";
$movecmd = "mv";
$rmcmd = "rm -f";
$verfycmd = "dscverify";
$verifysignatures = 0;
$verifysignaturesdistinput = 0;
$scinstall = "installed";
$distinputcriteria = "^.*\\.deb\\$";
$bzip = 1;
$gpgkey = "12345678";
$gpgpassfile = "$ENV{HOME}/.gnupg/passphrase";

%distinputdirs = (
    oldstable => 'oldstable',
    stable => 'stable',
    testing => 'testing',
    unstable => 'unstable',
    experimental => 'experimental'
);

@distributions = (
    'oldstable',
    'stable',
    'testing',
    'unstable',
    'experimental'
);

%distmapping = (
    oldstable => 'woody',
    stable => 'sarge',
    testing => 'etch',
    unstable => 'sid'
);

@architectures = ( 'i386' );

@sections = ( 'main', 'contrib', 'non-free' );

# @mailtos = (
#     'Maintainer',
#     'Uploaders',
#     '@bar.com',
#     'installer@foo.com'
# );

%release = (
    'origin' => "domain.tld",
    'label' => "domain.tld unofficial Repository",
    'description' => "unofficial packages from domain.tld"
);
```

## Konfigurationsdatei debarchiver.conf

Weitere Informationen können unter **man debarchiver** nachgelesen werden.

```
$destdir = "pfad";
```

Das Verzeichnis in dem das Repository generiert wird.

```
$inputdir = "pfad";
```

Das Verzeichnis, in dem nach neuen Paketen gesucht wird.

```
$copycmd = "befehl";
```

Das Kommandos zum Kopieren von Dateien.

```
$movecmd = "befehl";
```

Das Kommandos zum Verschieben von Dateien.

```
$rmcmd = "befehl";
```

Das Kommandos zum Löschen von Dateien.

```
$vrfycmd = "befehl";
```

Das Kommando, mit dem die Signatur von Paketen bzw. Paketdateien überprüft werden kann.

```
$verifysignatures = 0/1;
```

Legt fest, ob die Signatur von Paketen bzw. Paketdateien überprüft werden soll, die nach `$inputdir` kopiert wurden. 0 (*NULL*) deaktiviert und 1 aktiviert die Signaturprüfung.

```
$verifysignaturesdistinput = 0/1;
```

Legt fest, ob die Signatur von Paketen bzw. Paketdateien überprüft werden soll, die nach `%distinputdirs` kopiert wurden. 0 (*NULL*) deaktiviert und 1 aktiviert die Signaturprüfung.

```
$bzip = 0/1;
```

Legt fest, ob zusätzlich **bzip2**-komprimierte Index-Dateien (`Sources.bz2` und `Packages.bz2`) erstellt werden sollen. Diese Dateien werden **nicht** automatisch entfernt, sollte man sich später gegen diese Option entscheiden.

```
$scinstall = "verzeichnis";
```

Legt den Namen des Verzeichnisses fest, in das die `.changes` Dateien installiert werden: `$destdir/$distribution/$scinstall`

```
$distinputcriteria = "regex";
```

Ist ein Regex-Ausdruck <sup>[30]</sup> <sup>[31]</sup>, der auf alle Binary-Pakete zutreffen soll bzw. muss, die keine `.changes` Datei haben, aber in das Repository aufgenommen werden sollen (siehe auch `%distinputdirs`).

```
%distinputdirs = (distribution => 'verzeichnis');
```

Wie bereits unter [Abschnitt 3.1, „Struktur und Idee des Programmes“](#) beschrieben, können Binary-Pakete in spezielle Verzeichnisse unterhalb von `$inputdir` kopiert werden, damit diese, falls `$distinputcriteria` zutrifft, in das Repository aufgenommen werden. Mit dieser Option stellt man das Mapping ein, also z.B. den Verzeichnisnamen für Unstable-Pakete: `$inputdir/unstable`

```
@distributions = ('distribution');
```

Welche Distributionen (`$destdir/$distribution`) im Repository vorhanden sein sollen, z.B: *stable*, *testing*, *unstable*, *experimental*. Es ist wichtig, dass, falls ein Mapping verwendet werden soll, `%distmapping` zeitgleich gesetzt wird. Andernfalls erhält man 2 Verzeichnisse - eins für den Alias und eins für die Distribution.

```
%distmapping = (distribution => 'alias');
```

Distributionen bzw. Releases besitzen einen Alias. Die aktuelle »stable«-Distribution z.B. heißt »sarge«. debarchiver erstellt auf der Grundlage des Mappings einen Symlink `$destdir/$distribution -> $destdir/$alias`. Die Distributionen müssen zeitgleich in `@distributions` angegeben werden, damit `$destdir/$distribution` nicht als Verzeichnis angelegt wird.

```
@architectures = ('architektur');
```

Welche Architekturen zur Verfügung stehen sollen. Die Architekturen *all* und *source* werden automatisch angelegt. Weitere könnten sein: *alpha*, *amd64*, *arm*, *hppa*, *i386*, *ia64*, *m68k*, *mips*, *mipsel*, *powerpc*, *s390*, *sparc*

```
@sections = ('sektion');
```

Die Sektionen unterhalb von `@distributions`. Möglich sind: *main*, *contrib*, *non-free*

<sup>[30]</sup> [http://www.kore-nordmann.de/reg\\_exps\\_lernen.html](http://www.kore-nordmann.de/reg_exps_lernen.html)

<sup>[31]</sup> [http://www.regenechsen.de/regex\\_de/regex\\_1\\_de.html](http://www.regenechsen.de/regex_de/regex_1_de.html)

```
@mailto = ('feld'|@domain.tld|adresse@domain.tld');
```

Es ist möglich, dass debarchiver eine E-Mail an den Paket-Maintainer/Uploader oder eine definierte eMail-Adresse versendet, wenn ein Paket erfolgreich in das Repository aufgenommen wurde. Folgende Möglichkeiten existieren:

1. eMail-Adresse aus der `.changes` Datei auslesen: Dann muss die entsprechende Feldbezeichnung gewählt werden, z.B.: *Maintainer, Uploaders*
2. eMail-Adresse oder Domain explizit angeben. Für letzteres wird der Domain ein »@« vorangestellt.

```
%release = ('feld' => 'inhalt');
```

Die Release Dateien enthalten Informationen über das Repository. Die Felder *origin, label, description* können hier mit Inhalt gefüllt werden.

```
$cachedir = "pfad";
```

Das Verzeichnis, in dem die von **apt-ftparchive** erzeugten Cache-Dateien `cache.db` abgelegt werden.

```
$gpgkey = "gpg-schlüssel";
```

Der GPG-Schlüssel, der zum Signieren des Archivs, wie in [Abschnitt 3.5, „\[Optional\] Repository signieren“](#) beschrieben, benutzt werden soll.

```
$gpgpassfile = 0/"passwortdatei";
```

Die Datei, die das Passwort für `$gpgkey` enthält. Wird kein Passwort für `$gpgkey` benötigt, sollte dieser Wert auf `0 (NULL)` gesetzt werden. Standardmäßig wird das Passwort in `~/ .gnupg/passphrase` erwartet.

debarchiver startet direkt nach der Installation, da der Cron-Job standardmäßig alle 5 Minuten ausgeführt wird. Die aktuellen Konfigurationsdateien enthalten aber bereits das zukünftige (Sarge -> stable, Etch -> testing, Sid -> unstable) und nicht das aktuelle Mapping. Außerdem erstellt **debarchiver** standardmäßig ein recht umfangreiches Repository. Daher sollte man nach dem Erstellen der Konfigurationsdatei den Inhalt von `/var/lib/debarchiver/dists` noch einmal löschen. Nach dem nächsten Cron-Job steht dann die gewählte Struktur des Repository zur Verfügung.

```
# rm -r /var/lib/debarchiver/dists/*
```

Für ein Beispiel, wie ein solches Archiv später aussehen kann, kann man einen Blick in mein eigenes Repository werfen: <http://debian.wgdd.de/debian/>

## 3.3. Funktionsweise und Hinweise zur Nutzung

Nachdem debarchiver konfiguriert wurde, sollte man noch einiges über die Funktionsweise wissen. Standardmäßig wird **debarchiver** mit den Optionen `-so` aufgerufen. Die Option `-o|--addoverride` fügt `override` und `override.src` automatisch neue Einträge hinzu. Interessant ist die Option `-s|--scandetect`. Ist diese Option gesetzt, prüft **debarchiver** zuerst, ob **dpkg-scanpackages/dpkg-scansources** <sup>[15]</sup> und/oder **apt-ftparchive** <sup>[16]</sup> installiert sind und nutzt dann das installierte Paket. Sind beide Pakete installiert, wird **apt-ftparchive** aufgerufen. Die Arbeit mit beiden Frontends soll in [Abschnitt 3.3.1, „debarchiver und apt-ftparchive“](#) und [Abschnitt 3.3.2, „debarchiver und dpkg-scanpackages/dpkg-scansources“](#) näher erläutert werden.

### 3.3.1. debarchiver und apt-ftparchive

**apt-ftparchive** ist im Paket `apt-utils` enthalten:

```
# apt-get install apt-utils
```

Möchte man mit **debarchiver** und **apt-ftparchive** arbeiten, kann man statt `-s|--scandetect` explizit die Option `-x|--index` übergeben. **debarchiver** erzeugt während eines Laufs eine temporäre Konfigurationsdatei `$destdir/.apt-ftparchive.conf` für **apt-ftparchive** mit folgendem Inhalt:

## Beispiel 2. temporäre Konfigurationsdatei `.apt-ftparchive.conf`

```
Dir {
  ArchiveDir "$destdir";
  Cachedir "$cachedir";
};

Default {
  Packages::Compress ". gzip bzip2";
  Sources::Compress ". gzip bzip2";
};

TreeDefault {
  BinCacheDB "cache.db";
  Release::Origin "$release[origin]";
  Release::Label "$release[label]";
};

Tree "dists/distribution1" {
  Sections "sektion(en)";
  Architectures "architektur(en) all source";
  Release::Suite "distribution1";
  Release::Codename "alias1";
};

Tree "dists/distribution2" {
  Sections "sektion(en)";
  Architectures "architektur(en) all source";
  Release::Suite "distribution2";
  Release::Codename "alias2";
};

...
```

Da `apt-ftparchive` Pakete in `binary-all` nicht richtig handhaben kann, korrigiert `debarchiver` anschließend noch die Index-Dateien für alle Architekturen. Nur mit `apt-ftparchive` werden die für `apt-file` notwendigen Dateien `Contents-$arch(.gz)` kreiert.

### 3.3.2. debarchiver und dpkg-scanpackages/dpkg-scansources

`dpkg-scanpackages` und `dpkg-scansources` sind im Paket `dpkg-dev` enthalten:

```
# apt-get install dpkg-dev
```

Möchte man mit `debarchiver` und `dpkg-scanpackages/dpkg-scansources` arbeiten, kann man statt `-s|--scandetect` explizit die Option `-a|--autoscan` übergeben. `debarchiver` generiert dann alle Index-Dateien und zusätzlich `Package.gen` Dateien, die den Scanvorgang beschleunigen sollen. Diese Dateien werden nach einem Wechsel zu `apt-ftparchive` (siehe [Abschnitt 3.3.1, „debarchiver und apt-ftparchive“](#)) nicht automatisch gelöscht. Auch mit `dpkg-scanpackages/dpkg-scansources` werden alle Pakete in `binary-all` automatisch gehandhabt.

### 3.3.3. Sortierung der Pakete

Wie bereits in [Abschnitt 2.2, „Warum debarchiver?“](#) angesprochen, erstellt `debarchiver` keine Pool-Struktur. Allerdings werden nicht alle Pakete in einem Verzeichnis zusammengefasst, sondern entsprechend ihrer Untersektion <sup>[32]</sup> in entsprechende Unterverzeichnisse sortiert. Wie diese Struktur funktioniert, soll kurz an einem Beispiel erläutert werden. Gegeben sei ein Paket `foobar`, zugehörig zu `Sektion/Untersektion` (z.B. `contrib/science`): die Paketdateien befinden sich dann entsprechend in `$destdir/$distribution/$sektion/binary-$arch/$untersektion` und die Quellen in `$destdir/$distribution/$sektion/source/$untersektion`.

Allerdings ist angedacht, die Pakete später in einem Pool zu organisieren.

<sup>[32]</sup> <http://www.debian.org/doc/maint-guide/ch-dreq.de.html#s-control>

## 3.4. [Optional] Signaturen mit GnuPG verifizieren

Es ist möglich, dass debarchiver die GnuPG-Signaturen von Paketen überprüft und nur Pakete, deren Signatur verifiziert werden kann, in das Repository kopiert. Diese Einstellung kann separat für `$inputdir` (`$verifysignatures`) und `%distinputdirs` (`$verifysignaturesdistinput`) getätigt werden. Um diese Möglichkeit zu nutzen, werden folgende Schritte ausgeführt:

### Prozedur 1. GnuPG-Verifizierung ermöglichen

#### 1. [Optional] Verändern des debarchiver-Heimatverzeichnisses

Dieser Schritt ist wirklich optional. Er basiert auf der Überlegung, dass durch spätere Schritte ([Abschnitt 4.5](#), „[Optional] Pakete über `http://` verfügbar machen“, [Abschnitt 4.6](#), „[Optional] Eine Übersicht aller vorhandenen Pakete (on-the-fly) erstellen“) das Heimatverzeichnis des Benutzers `debarchiver` (`~debarchiver/`) und damit die in diesem Verzeichnis liegenden System- und Konfigurationsdateien - die in den nächsten Schritten erzeugt werden - öffentlich zugänglich sind. Der Zugriff auf diese Dateien und Ordner kann auch über die Konfigurationsdatei des Apache-Webserver [Beispiel 7](#), „`/etc/apache/conf.d/debarchiver`“ eingeschränkt werden. Es gibt mehrere Möglichkeiten, das Problem zu umgehen. Ich bevorzuge die strikte Trennung.

```
# mkdir /home/debarchiver
# chown debarchiver.debarchiver /home/debarchiver
# usermod -d /home/debarchiver debarchiver
```

#### 2. GnuPG-Infrastruktur anlegen und konfigurieren

- a. Zuerst geben wir dem Konto »debarchiver« Zugriff auf eine echte Shell und loggen uns ein:

```
# chsh -s /bin/bash debarchiver
# su debarchiver // oder als der Benutzer, der das Repository unterhält
$ cd ~
```

- b. Es ist notwendig die Infrastruktur für GnuPG zu erzeugen:

```
$ echo -n "" | gpg
$ vi .gnupg/gpg.conf // Alle Schlüssel-Server auskommentieren (optional)
// speichern und schließen
```

- c. Nun müssen/können die Schlüssel der Uploader importiert werden:

```
$ gpg --no-default-keyring --keyring uploaders.gpg --import Schlüssel-Datei
// oder von einem Schlüssel-Server
$ gpg --no-default-keyring --keyring uploaders.gpg --keyserver Schlüssel-Server
--recv-keys Schlüssel-ID
```

- d. Der Schlüsselring `uploaders.gpg` muss während des Ausführens von `dscverify` bekannt sein:

```
$ cp /etc/devscripts.conf ~/.devscripts
$ vi .devscripts
// Kommentarzeichen vor DSCVERIFY_KEYRINGS=""
// entfernen und Schlüsselring wie beschrieben hinzufügen:
[.]
DSCVERIFY_KEYRINGS=~/.gnupg/uploaders.gpg"
[.]
// speichern und schließen
```

- e. Wer ein signiertes Paket parat hat, kann jetzt testen, ob alles funktioniert:

```
$ dscverify package.changes
package.changes:
  Good signature found
validating package.dsc
  Good signature found
validating package.orig.tar.gz
validating package.diff.gz
validating package.deb
All files validated successfully.
```

### 3. Signatur-Prüfung aktivieren

- a. Zunächst entzieht man dem Konto »debarchiver« wieder den Zugriff auf die Shell:

```
$ exit
# chsh -s /bin/false debarchiver
```

Wer will, kann nun noch die nicht länger benötigten Dateien löschen:

```
# rm ~debarchiver/.bash_history ~debarchiver/.viminfo // optional
```

- b. Zum Schluss aktiviert man die Prüfung der Signaturen in `debarchiver.conf` (`$vrfycmd`, `$verifysignatures`):

```
# vi /etc/debarchiver.conf
// entsprechend Konfigurationsdatei debarchiver.conf
[.]
$vrfycmd = "dscverify";
$verifysignatures = 1;
[.]
// speichern und schließen
```

Damit ist die Signaturprüfung aktiv.

Falls das Heimatverzeichnis des Benutzers »debarchiver« verändert wurde, dann ist möglich, dass trotz erfolgreichen Testlaufs (Schritt 2.e), die Signatur-Prüfung fehlschlägt, weil `/var/lib/debarchiver/.gnupg/secring.gpg` nicht gefunden werden kann. Dieser Fehler ist mir unverständlich. In diesem Fall hilft jedoch:

```
# usermod -d /var/lib/debarchiver debarchiver
# usermod -d /home/debarchiver debarchiver
```

Nun sollte die Signatur-Prüfung funktionieren.

## 3.5. [Optional] Repository signieren

Mit der Veröffentlichung von apt 0.6 wird erstmalig eine Integritätsprüfung für Repositories eingeführt. Der Grundgedanke und die Funktionsweise werden in der *Anleitung zum Absichern von Debian* <sup>[33]</sup> und in der Dokumentation zu apt-secure <sup>[34]</sup> näher erläutert. Um unser Archiv dafür entsprechend zu präparieren, sind nur wenige Schritte notwendig.

<sup>[33]</sup> <http://www.debian.org/doc/manuals/securing-debian-howto/ch7#s7.4.1>

<sup>[34]</sup> `man apt-secure`

## Prozedur 2. Repository mit GnuPG signieren

### 1. [Optional] Verändern des debarchiver-Heimatverzeichnisses

Siehe [Schritt 1](#) in [GnuPG-Verifizierung ermöglichen](#).

### 2. GnuPG-Infrastruktur anlegen und Schlüssel erstellen

- Siehe [Schritt 2.a](#) und [Schritt 2.b](#) (nur *einmalig(!)* notwendig, wenn die Infrastruktur noch nicht existiert) in [GnuPG-Verifizierung ermöglichen](#).
- Im nächsten Schritt erzeugen wir einen primären Schlüssel, der jedoch nicht dazu benutzt werden sollte, das Repository zu signieren. Dieser wird später in [Schritt 2.c](#) erzeugt.

```
$ gpg --gen-key
```

- Nun legen wir den eigentlichen Archivschlüssel an:

```
$ gpg --gen-key
```

- Da **debarchiver** ohne Nutzer-Interaktion läuft, muss das Passwort - sofern eines gewählt wurde - hinterlegt werden (das sollte man mit Passwörtern normalerweise natürlich nicht tun). Wir nutzen dazu standardmäßig `~/.gnupg/passphrase`. Es kann aber auch eine beliebige andere Datei sein. Nur sollte man immer auf die Zugriffsberechtigungen achten, damit diese Datei für andere Systembenutzer nicht lesbar ist:

```
$ touch ~/.gnupg/passphrase
$ chmod 600 ~/.gnupg/passphrase
$ vi ~/.gnupg/passphrase // Passwort eingeben
// speichern und schließen
```

- Damit andere die Integrität unseres Archivs überprüfen können, muss der Archiv-Schlüssel zugänglich gemacht werden. Dazu exportiert man ihn entweder (im ASCII-Format) als Datei oder hinterlegt ihn auf einem öffentlichen Server:

```
$ gpg --armor --output archiv_schluesssel.asc --export Schlüssel-ID
$ gpg --keyserver Schlüssel-Server --send-key Schlüssel-ID
```

Will man das Repository nur privat und lokal nutzen, ist ersteres ausreichend. Mehr später in [Abschnitt 4.4](#), „Integrität des Repository verifizieren“.

### 3. Signieren des Archivs aktivieren

- Siehe [Schritt 3.a](#) in [GnuPG-Verifizierung ermöglichen](#).
- Zum Schluss aktiviert man das Signieren des Archivs bzw. der Release Dateien in `debarchiver.conf` (`$gpgkey`, `$gpgpassfile`):

```
# vi /etc/debarchiver.conf
// entsprechend Konfigurationsdatei debarchiver.conf
[.]
$gpgkey = "Schlüssel-ID";
[.]
$gpgpassfile = "${ENV{HOME}}/.gnupg/passphrase"; // (Passwort erforderlich)
$gpgpassfile = 0; // (Passwort nicht erforderlich)
[.]
// speichern und schließen
```

Damit werden signierte `Release.gpg` Dateien beim nächsten Lauf von **debarchiver** erstellt.

## 4. Arbeiten mit dem Repository

Nachdem das Repository erstellt und konfiguriert wurde, können nun die ersten Pakete hinzugefügt werden. Neben der reinen Pflege des Repositories, sind in diesem Abschnitt auch einige »Goodies« zu finden, mit denen man das Repository schmücken kann.

### 4.1. Pakete zum Repository hinzufügen

Um den Prozess des Uploads zu vereinfachen, können wir `dput`<sup>[35]</sup> nutzen. Dieses muss, falls noch nicht getan, ebenfalls installiert werden:

```
# apt-get install dput
```

Als Alternative zu `dput` existiert `dupload`<sup>[36]</sup>, das ebenfalls genutzt werden kann. Ausführliche Informationen zu diesen Programmen findet man in den Man-pages<sup>[37]</sup>.



#### NON-Maintainer-Uploads (NMU) und Backports

Wer in seinem Repository NMU-Pakete oder Backports zur Verfügung stellen will, dem sei hiermit noch der Hinweis auf die `dpkg-buildpackage` Option `-sa` auf den Weg gegeben (siehe auch `man debuild`). Damit wird auch das Quellpaket (nur für `.orig.tar.gz` relevant) in `.changes` eingetragen und bei einem Upload ebenfalls übertragen. Das ist wichtig, da Backports selten die Version `-1` tragen und NMU-Pakete dies i.A. ebenfalls nicht tun (sollten). Weiterhin ist es wichtig, da sich auch `debarchiver` an der `.changes` orientiert und ohne diese Vorgehensweise die Quellen nicht ein- bzw. zurodnen könnte (siehe auch Abschnitt 6.5 im *Debian New Maintainers' Guide*<sup>[3]</sup>).

Es wird von `debarchiver` nicht geprüft, ob das Quellpaket mit hochgeladen wird/wurde.

#### 4.1.1. Ein lokales Repository bedienen

Für ein lokales Archiv fügt man `~/dput.cf` oder `/etc/dput.cf` einfach folgenden Eintrag hinzu:



#### Kurzbezeichnung für den Server

Die gewählte Kurzbezeichnung für den Server (innerhalb der eckigen Klammern) muss eindeutig sein. Es darf keinen zweiten Eintrag mit der selben Bezeichnung in `~/dput.cf` oder `/etc/dput.cf` vorhanden sein. Die hier gewählte Bezeichnung `[lokal]` sollte nicht mit einer neuen Installation von `dput` kollidieren.

#### Beispiel 3. Eintrag in `dput.cf` für ein lokales Archiv

```
[lokal]
fqdn = localhost
method = local
incoming = /var/lib/debarchiver/incoming
allow_unsigned_uploads = 0
```

Der letzte Eintrag garantiert, dass nur signierte Pakete mit `dput` übertragen werden. Falls die Paket-Signaturen, wie in [Abschnitt 3.4, „\[Optional\] Signaturen mit GnuPG verifizieren“](#) beschrieben, überprüft werden, ist es dringend empfohlen, den Upload von unsignierten Paketen auf diesem Weg zu verbieten.

<sup>[35]</sup><http://packages.debian.org/dput>

<sup>[36]</sup><http://packages.debian.org/dupload>

<sup>[37]</sup>`man dput | dput.cf | dupload | dupload.conf`

Um nun ein Paket in das Repository zu übertragen, reicht ein:

```
$ dput lokal package.changes
```

Das Paket liegt nun in `/var/lib/debarchiver/incoming` (`$inputdir`, siehe zusätzlich Hinweis in `.changes` und `$inputdir` bzw. `%distinputdirs`) und wird beim nächsten Lauf von **debarchiver** in das Repository übertragen.

Pakete, für die keine `.changes` Datei existiert (z.B. Kernelpakete oder Module), werden einfach in das jeweilige Distributionsverzeichnis kopiert (siehe auch [Abschnitt 3.1, „Struktur und Idee des Programmes“](#) und `%distinputdirs`):

```
$ cp package.deb /var/lib/debarchiver/incoming/$distribution/
```

## 4.1.2. Ein entferntes Repository bedienen

Für ein entferntes Archiv (das Repository befindet sich nicht auf dem lokalen System) ist der Eintrag in `~/.dput.cf` oder `/etc/dput.cf` von der Übertragungsmethode abhängig. Möglich sind z.B. **ftp**, **scp** oder auch **rsync**. Ich möchte an dieser Stelle nur 2 Beispiele aufführen. Weitere Informationen, auch zu den in allen Beispielen genannten Optionen, findet man in den Man-pages <sup>[37]</sup>.

### Beispiel 4. Eintrag in dput.cf für einen FTP-Server mit Login und Nutzung von passivem FTP, wobei alle Pakete in Testing einsortiert werden sollen

```
[ftpserver]
fqdn = ftp.domain.tld
method = ftp
login = user
incoming = /var/lib/debarchiver/incoming/testing
allow_unsigned_uploads = 0
passive_ftp = 1
```

### Beispiel 5. Eintrag in dput.cf für einen Server mit SSH-Authentifizierung über Publickey (der SSH-Server lauscht auf Port 2222 und /home/user/.ssh/id\_dsa enthält den notwendigen Schlüssel zur Authentifizierung)

```
[scpserver]
fqdn = domain.tld
login = user
method = scp
ssh_config_options = Host=domain.tld
IdentityFile2 /home/user/.ssh/id_dsa
Port=2222
incoming = /var/lib/debarchiver/incoming
allow_unsigned_uploads = 0
```

Pakete können dann sehr einfach mittels

```
$ dput ftpserver package.changes
$ dput scpserver package.changes
```

in das Repository übertragen werden. Nach dem nächsten Lauf von **debarchiver** stehen diese Pakete zum Download bereit. Um Pakete ohne eine `.changes` zu übertragen, kann natürlich ebenfalls die präferierte Methode (**ftp**, **scp**, **rsync**) genutzt werden.

## 4.2. Pakete aus dem Repository entfernen

Pakete können grundsätzlich nur per Hand aus dem Repository entfernt werden. Dazu muss man wissen, dass **debarchiver**, wenn ein Paket zwischen den Sektionen (z.B. von *contrib* nach *main*) oder Architekturen verschoben wird oder neue Upstream-Versionen in das Repository übertragen werden, die alten Dateien nicht löscht (vgl. BTS-Eintrag #119173 <sup>[38]</sup>). Dadurch sammeln sich im Laufe der Zeit möglicherweise einige Altlasten an. In diesem Fall kann bzw. sollte man das Repository selbst aufräumen. Dazu empfiehlt sich der folgende Weg:

```
# cd /var/lib/debarchiver/dists
# find . -name package*
// nun veraltete oder obsolete Dateien löschen
// danach evtl. Einträge in override und override.src löschen
```

Allerdings stellt sich nun das Problem, wie man die *Index*-Dateien aktualisiert. Dazu drei mögliche Wege:

1. Option `--scanall`



### Empfehlung

Diese Variante ist die sauberste von allen und kann nur empfohlen werden.

Damit werden alle Sektionen und Distributionen gescannt und die *Index*-Dateien neu erstellt. Diese Option funktioniert bei der Nutzung von **apt-ftparchive** etwas anders als mit **dpkg-scanpackages/dpkg-scansources**. Mit der erstgenannten Anwendung werden die *Index*-Dateien nur dann aktualisiert, wenn sich etwas verändert hat. Mit den beiden letztgenannten Anwendungen, werden die *Index*-Dateien bei jedem Lauf neu generiert.

2. Das Löschen von Paketen wird generell vor dem Upload eines neuen Pakets in die selbe Distribution erledigt. Nach dem Einsortieren des neuen Paktes stehen dann aktualisierte *Index*-Dateien innerhalb der Sektion/Distribution zur Verfügung.
3. Man erstellt sich ein kleines Paket (z.B. »test« - der Name sollte besser individueller gewählt werden), das bis zu 3 Paketen zur Verfügung stellt (z.B. test-main, test-contrib, test-nonfree) und lädt dieses Paket nach jedem Aufräumen in das Repository.

## 4.3. Pakete apt-get(ten)

Nachdem das Repository steht, fügt man `/etc/apt/sources.list` noch die folgenden Einträge hinzu und kann dann Pakete mittels **apt-get**, **aptitude** oder **dselect** installieren:

### Beispiel 6. /etc/apt/sources.list Einträge (file:/)

```
# nach der Distribution
deb file:/var/lib/debarchiver oldstable main contrib non-free
deb file:/var/lib/debarchiver stable main contrib non-free
deb file:/var/lib/debarchiver testing main contrib non-free
deb file:/var/lib/debarchiver unstable main contrib non-free
deb file:/var/lib/debarchiver experimental main contrib non-free

# nach dem Codenamen
deb file:/var/lib/debarchiver woody main contrib non-free
deb file:/var/lib/debarchiver sarge main contrib non-free
deb file:/var/lib/debarchiver etch main contrib non-free
deb file:/var/lib/debarchiver sid main contrib non-free
```

<sup>[38]</sup> <http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=119173>

Nun nur noch **apt-get update** und wahlweise **apt-file update** und die Pakete im Repository stehen zur Verfügung und können installiert werden.

```
# apt-get update && apt-file update
```

## 4.4. Integrität des Repository verifizieren

Wie in [Abschnitt 3.5](#), „[Optional] Repository signieren“ beschrieben, kann man die Integrität eines Archivs überprüfen. Dazu ist es notwendig, den Archiv-Schlüssel in den apt-Schlüsselring zu importieren. Dafür steht das Frontend **apt-key** zur Verfügung. Hat man den Schlüssel, wie in [Schritt 2.e](#) von [Repository mit GnuPG signieren](#) beschrieben, als Datei exportiert, dann importieren wir ihn ganz einfach mittels:

```
# apt-key add archiv_schluesel.asc
```



### apt-secure

Ohne diesen Schritt markiert aptitude signierte Archive als *untrusted source* (nicht vertrauenswürdige Quelle) und auch apt-get installiert Pakete aus dieser Quelle nur auf explizite Nachfrage (es sei denn, man hat mit `APT::Get::AllowUnauthenticated 1;` in `/etc/apt/apt.conf` nachgeholfen).

## 4.5. [Optional] Pakete über http:// verfügbar machen

Wer ein Repository unterhält, wird wahrscheinlich über kurz oder lang die Pakete anderen Debianern zur Verfügung stellen wollen. Alternativ möchte man das Repository vielleicht auch nur lokal (localhost oder LAN) per Web-Server zugänglich machen. Für den Apache wird dazu `/etc/apache/http.conf` oder `vhosts.conf` um die folgenden Einträge ergänzt. Alternativ kann man natürlich auch eine Datei `/etc/apache/conf.d/debarchiver` anlegen:

### Beispiel 7. /etc/apache/conf.d/debarchiver

```
Alias /debian /var/lib/debarchiver

<Directory "/var/lib/debarchiver/">
  Options Indexes FollowSymLinks +Includes
  # HeaderName /repHEADER # siehe mod_autoindex
  # ReadmeName /repREADME # siehe mod_autoindex

  # verbiete Zugriff auf Benutzer-Dateien
  <Files ~ "^\. (devscripts|viminfo|bash.*)$" >
    AllowOverride None
    Order deny,allow
    Deny from all
  </Files>

  AllowOverride None
  Order allow,deny
  Allow from all
</Directory>

# verbiete Zugriff auf .gnupg
<Directory /var/lib/debarchiver/.gnupg></Directory>
  AllowOverride None
  Order deny,allow
  Deny from all
</Directory>
```

In unserem Fall können dann die folgenden Einträge in der `/etc/apt/sources.list` ergänzt werden:

### Beispiel 8. `/etc/apt/sources.list` Einträge ([http://](#))

```
deb      http://domain.tld/debian woody main contrib non-free
deb-src  http://domain.tld/debian woody main contrib non-free
deb      http://domain.tld/debian oldstable main contrib non-free
deb-src  http://domain.tld/debian oldstable main contrib non-free

deb      http://domain.tld/debian sarge main contrib non-free
deb-src  http://domain.tld/debian sarge main contrib non-free
deb      http://domain.tld/debian stable main contrib non-free
deb-src  http://domain.tld/debian stable main contrib non-free

deb      http://domain.tld/debian etch main contrib non-free
deb-src  http://domain.tld/debian etch main contrib non-free
deb      http://domain.tld/debian testing main contrib non-free
deb-src  http://domain.tld/debian testing main contrib non-free

deb      http://domain.tld/debian sid main contrib non-free
deb-src  http://domain.tld/debian sid main contrib non-free
deb      http://domain.tld/debian unstable main contrib non-free
deb-src  http://domain.tld/debian unstable main contrib non-free
```

Nun nur noch **apt-get update** und wahlweise **apt-file update** und die Pakete im Repository stehen zur Verfügung und können installiert werden.

```
# apt-get update && apt-file update
```

## 4.6. [Optional] Eine Übersicht aller vorhandenen Pakete (on-the-fly) erstellen

Mit Hilfe eines kleinen PHP-Skripts [parse-apt-files.inc](#) von Jarno Elonen <sup>[10]</sup> ist es möglich, die Repository-Dateien `Packages.gz` und `Sources.gz` zu durchsuchen und die Informationen zu den Paketen automatisch aufzubereiten. Alle gefundenen Pakete und Informationen werden auf einer Webseite (inkl. Links zu den Paketen, Quellen und `.changes` Dateien, Paketbeschreibung, aktuell im Repository erhältliche Version, ...) zusammen gefasst. Mit einigen kleinen Änderungen (s.o., vor allem in Bezug auf den Pfad für die `.changes`-Dateien) arbeitet das Skript auch sehr schön mit von debarchiver generierten Repositories.

Ich empfehle die Datei `parse-apt-files.inc` oberhalb von `$destdir` (siehe [Beispiel 1](#), „Konfigurationsdatei für debarchiver“) abzulegen, da sonst `opendir()` ausgehebelt wird. Ein einfaches Beispiel, um sich die Pakete in *Sid (Unstable)* anzeigen zu lassen (mit der gefixten [parse-apt-files.inc](#)):

## Beispiel 9. Beispiel für eine Webseite index.php mit parse-apt-files.inc

```
<? echo "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de" lang="de">
<head>
  <title>Debian-Pakete</title>
</head>

<body>

<?php
// notwendig um das Skript einzubinden
include_once("parse-apt-files.inc");
// parse-apt-files.inc liegt wie index.php in /var/lib/debarchiver

// wir erstellen eine Paketliste für Sid (unstable)
echo "<h2>Pakete in Sid (unstable)</h2>";
parse_and_list(
  Array( "dists/sid/main/binary-i386/Packages.gz",
        "dists/sid/contrib/binary-i386/Packages.gz",
        "dists/sid/non-free/binary-i386/Packages.gz" ),
  Array( "dists/sid/main/source/Sources.gz",
        "dists/sid/contrib/source/Sources.gz",
        "dists/sid/non-free/source/Sources.gz" ),
  "dists/sid/installed" );
?>

</body>
</html>
```

Der Output könnte dann zum Beispiel [so](#) aussehen.

## 4.7. [Optional] Mehr ...

Weitere Ideen? Dann schreibt: [Abschnitt 1.5](#), „Feedback“