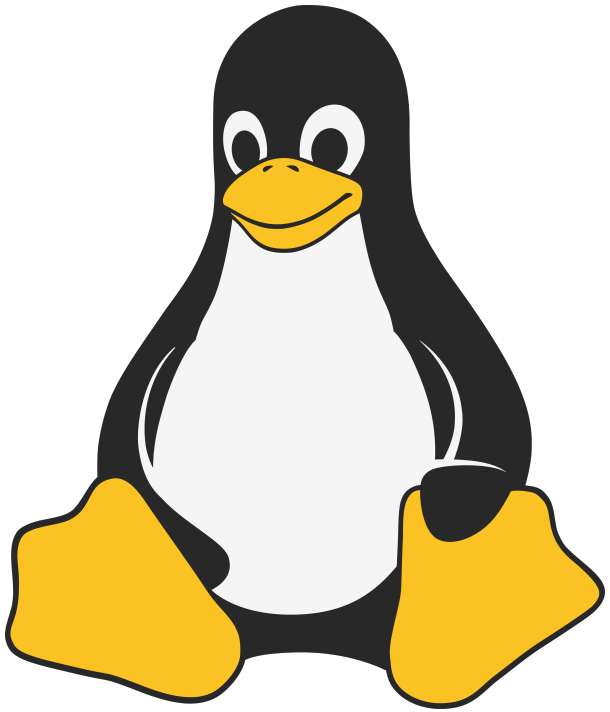
Paket-Verwaltung mit `apt` und `dpkg`



Inhaltsverzeichnis

- Pakete auflisten: `apt list`
- Pakete aktualisieren: `apt upgrade`
- Pakete installieren: `apt install <package>`
- Pakete entfernen
  - Pakete entfernen: `apt remove <package>`
  - Pakete entfernen incl. Konfigurationsdateien: `apt purge <package>`
  - Überflüssige Pakete entfernen: `apt autoremove`

- Paket-Abhängigkeiten
  - Abhängigkeiten anzeigen: `apt depends <package>`
  - Rekursive Abhängigkeiten anzeigen: `apt rdepends <package>`
- Paket-Quellen
  - `sources.list` und `sources.list.d/*.list`
  - Hinzufügen und Entfernen von Paket-Quellen
  - Vor- und Nachteile zusätzlicher Paket-Quellen
  - Empfehlungen zu zusätzlichen Paket-Quellen

Distributionen mit `apt`-Paket-Verwaltung

`apt` ist das Paket-Verwaltungssystem für Debian Linux und seine vielen Derivate, wie z.B. Ubuntu, Linux Mint, elementary OS, Rasberry Pi OS, Kali Linux, etc. Damit gewinnt es seine Bedeutung weit über Debian hinaus.

`apt` vs. `apt-get/apt-cache` vs. `aptitude`

- *`apt-get`* und *`apt-cache`*: sind die älteren Low-Level-Werkzeuge für die Kommandozeile, die immer noch unterstützt und gepflegt werden. Sie sind die Basis für `apt`.
- *`apt`*: ist ein neueres Kommandozeilen-Werkzeug, das die Funktionalität von `apt-get` und `apt-cache` in einem einzigen Befehl vereint und erweitert. Es ist einfacher zu bedienen und bietet eine bessere Benutzererfahrung.
- *`aptitude`*: ist ein weiteres Paket-Verwaltungswerkzeug, das auf `apt` basiert. Es ist ein interaktives Werkzeug mit einer Text-basierten Benutzeroberfläche.

In diesem Tutorial ignorieren wir `apt-get`, `apt-cache` und `aptitude` und konzentrieren uns auf `apt` und ein wenig auch auf `dpkg`.

`apt` und `dpkg` im Zusammenspiel

- *`dpkg`*: ist der Low-Level-Paket-Manager zur Installation, Konfiguration und Deinstallation einzelner Pakete. `dpkg` arbeitet direkt mit den `.deb`-Paketen. Es kennt keine Paket-Abhängigkeiten.
- *`apt`*: ist ein Frontend für `dpkg`. `apt` hat sozuagen das gesamte Paket-System im Blick. Es kann **transitive Paket-Abhängigkeiten** auflösen und die benötigten Pakete automatisch mitinstallieren. (Paket A benötigt Paket B und Paket B benötigt Paket C. `apt` installiert dann alle drei Pakete, wenn Paket A installiert wird.)

In der Praxis der Paket-Verwaltung wird `dpkg` nur selten direkt verwendet. Am ehesten wird es verwendet, um Paket-Informationen anzuzeigen, Paket-Inhalte aufzulisten oder um installierte Pakete nach Dateien zu durchsuchen. Also eher für lesende Zugriffe auf die Paket-Datenbank. Für ändernde Zugriffe ist `dpkg` eher ungeeignet, da es keine Abhängigkeiten berücksichtigen kann.

Das Arbeitspferd der Paket-Verwaltung ist `apt`. Es ist das Werkzeug, das für die Installation, Aktualisierung und Entfernung von Paketen verwendet wird.

In diesem Tutorial steht `apt` im Vordergrund.

Superuser Privileges

Die Paket-Verwaltungs-Kommandos können mit normalen Benutzerrechten ausgeführt werden, wenn sie nur lesende Zugriffe auf das Paket-System durchführen (Abfragen, Anzeigen, Suchen, etc.).

Für ändernde Zugriffe (Installation, Aktualisierung, Entfernung, etc.) sind Superuser-Rechte erforderlich. Das bedeutet, dass die Kommandos mit `sudo` oder in einer `root`-Shell auszuführen sind.

Debian-Paket-Format

Debian-Pakete sind Archivdateien mit der Dateiendung `.deb`, die alle für die Installation des Pakets erforderlichen Dateien enthalten. Ein `deb`-Paket enthält in einem definierten Format (hier verkürzt):

- Metadaten über das Paket (Name, Version, Abhängigkeiten, Maintainer, etc.)
- Skripte, die vor und nach der Installation bzw. Deinstallation des Pakets ausgeführt werden
- die Dateien des Pakets, die auf dem Zielsystem installiert werden: Programme, Bibliotheken, Konfigurationsdateien, Manpages, etc.

`dpkg` ist das Werkzeug, das direkt mit den `.deb`-Paketen arbeitet. Es kann Pakete installieren, deinstallieren, konfigurieren, Informationen über Pakete anzeigen und Paket-Inhalte auflisten.

Da `dpkg` Abhängigkeiten nicht berücksichtigt, arbeiten wir nicht direkt mit `dpkg`, sondern mit `apt`. `apt` ist ein Frontend für `dpkg`.

Paket-Quellen

`apt` bezieht die Paket-Informationen und die `.deb`-Pakete von Paket-Quellen. Zu Beginn der Installation verweist die Paket-Quelle zunächst auf das Installationsmedium (DVD, USB-Stick, ISO-Image). Nach der Installation des Basis-Systems werden die Paket-Quellen auf die Online-Server von Debian umgestellt. Dies sind Server, auf denen die Paket-Informationen und die `.deb`-Pakete gespeichert sind. Die Paket-Quellen sind in der Datei `/etc/apt/sources.list` und in den Dateien im Verzeichnis `/etc/apt/sources.list.d/` konfiguriert.

Mirrors oder Spiegel-Server

Der Standard-Server für Debian-Pakete ist `http://deb.debian.org/debian`. In vielen Ländern gibt es Spiegel-Server, die die Pakete von Debian spiegeln (in Kopie vorhalten). Der Spiegel-Server sollte im weltweiten Netzwerk so gewählt werden, dass er "möglichst nah" am eigenen Standort lieget, um die Pakete schnell herunterladen zu können. Außerdem entlastet die Nutzung von Spiegel-Servern den zentralen Debian-Server.

Paket-Metadaten und der Installations-Prozess

Der Installations-Prozess läuft grundsätzlich in zwei Schritten ab:

- **Paket-Metadaten aktualisieren**: `apt` holt sich die aktuellen Paket-Informationen von den Paket-Quellen (`apt update`). Diese Informationen enthalten die Metadaten aller verfügbaren Pakete: Name, Version, Abhängigkeiten, etc. Liegen die Metadaten vor, kann `apt` anhand der Versionsnummern prüfen, ob es Aktualisierungen für installierte Pakete gibt.

- **Pakete aktualisieren oder installieren**: Anhand der Metadaten kann `apt` feststellen, welche Pakete aktualisiert werden müssen (`apt upgrade`) oder welche Version eines Pakets bei einer Neuinstallation installiert werden soll (`apt install`).

Da sich mit einer neuen Paket-Version auch die Abhängigkeiten ändern können, kann es vorkommen, dass `apt` zusätzliche Pakete installieren muss und andere Pakete entfernen kann, um die Abhängigkeiten zu erfüllen.

Paket-Metadaten aktualisieren: `apt update`

```
$ sudo apt update
```

Dieses Kommando aktualisiert die Paket-Informationen aus den Paket-Quellen. Es holt sich die Metadaten aller verfügbaren Pakete und speichert sie in der lokalen Paket-Datenbank.

Pakete auflisten: `apt list`

```
$ apt list <pattern>
```

Mit diesem Kommando können alle verfügbaren Pakete oder nur die Pakete, die dem Namensmuster entsprechen, aufgelistet werden.

```
$ apt list --installed
```

Mit der Option `--installed` werden nur die installierten Pakete aufgelistet.

```
$ apt list --upgradable
```

Mit der Option `--upgradable` werden nur die Pakete aufgelistet, für die es Aktualisierungen gibt.

Pakete aktualisieren: `apt upgrade`

```
$ sudo apt upgrade
```

Mit diesem Kommando werden alle installierten Pakete auf den neuesten Stand gebracht. `apt` prüft die Metadaten und aktualisiert nur die Pakete, für die es eine neuere Version gibt (also die Pakete, die mit `apt list --upgradable` aufgelistet werden).

Pakete installieren: `apt install <package>`

```
$ sudo apt install curl
```

Mit diesem Kommando wird das angegebene Paket (hier `curl`) installiert. `apt` prüft die Abhängigkeiten und installiert die benötigten Pakete automatisch mit. Dabei installiert `apt` die Versionen der Pakete, die in den Metadaten angegeben sind, die beim letzten `apt update` heruntergeladen wurden.

Pakete entfernen

Beim Entfernen von Paketen gibt es verschiedene Möglichkeiten:

- Paket entfernen und die Konfigurationsdateien behalten
- Paket mit allen Konfigurationsdateien entfernen
- Überflüssige Pakete entfernen

Pakete entfernen: `apt remove <package>`

```
$ sudo apt remove curl
```

Mit diesem Kommando wird das angegebene Paket (hier `curl`) deinstalliert. Die Konfigurationsdateien des Pakets bleiben erhalten.

Pakete entfernen incl. Konfigurationsdateien:

`apt purge <package>`

```
$ sudo apt purge curl
```

Mit diesem Kommando wird das angegebene Paket (hier `curl`) deinstalliert und die Konfigurationsdateien des Pakets werden ebenfalls entfernt.

Überflüssige Pakete entfernen: `apt autoremove`

```
$ sudo apt autoremove
```

Mit diesem Kommando werden alle Pakete entfernt, die installiert wurden, um Abhängigkeiten zu erfüllen, aber jetzt nicht mehr benötigt werden. Das sind z.B. Pakete, von denen andere Pakete abhingen, die aber inzwischen deinstalliert wurden.

Paket-Cache

`apt` speichert die heruntergeladenen `.deb`-Pakete in einem lokalen Paket-Cache. Dieser Cache kann mit der Zeit anwachsen und benötigt Speicherplatz. Der Cache kann...

- von überflüssigen Paketen bereinigt oder auch
- komplett geleert werden.

Paket-Cache um überflüssige Pakete bereinigen:

`apt autoclean`

```
$ sudo apt autoclean
```

Mit diesem Kommando werden nur die veralteten `.deb`-Pakete aus dem Cache entfernt. Die `.deb`-Pakete, die noch benötigt werden, bleiben erhalten.

Paket-Cache komplett bereinigen: `apt clean`

```
$ sudo apt clean
```

Mit diesem Kommando werden alle `.deb`-Pakete aus dem Cache entfernt. Der Cache wird komplett geleert.

Paket-Informationen und Inhalt

- `dpkg` kann nur mit lokal vorliegenden `.deb`-Paketen arbeiten. Es kann deshalb nur über installierte Pakete Auskunft geben.
- `apt` kann die Paket-Informationen aus den heruntergeladenen Paket-Metadaten beziehen. Es kann deshalb auch über nicht installierte Pakete Auskunft geben. Allerdings kennt `apt` nur die Metadaten, nicht den Inhalt der Pakete. Es kann deshalb nicht über die in den Paketen enthaltenen Dateien Auskunft geben.
- `apt-file` ist ein zusätzliches Werkzeug, das den Inhalt der Pakete kennt und auch über die enthaltenen Dateien Auskunft geben kann.

Paket-Informationen anzeigen: `apt show`

Mit diesem Kommando werden die Metadaten des angegebenen Pakets (hier `curl`) angezeigt. Zu den Metadaten gehören:

- Name
- Version
- Beschreibung
- Abhängigkeiten
- Maintainer
- Homepage
- etc.

```
$ apt show curl
Package: curl
Version: 7.88.1-10+deb12u8
Priority: optional
Section: web
Maintainer: Alessandro Ghedini <ghedo@debian.org>
Installed-Size: 500 kB
Depends: libc6 (>= 2.34), libcurl4 (= 7.88.1-10+deb12u8), zlib1g (>= 1:1.1.4)
Homepage: https://curl.se/
Tag: implemented-in::c, interface::commandline, network::client,
 protocol::ftp, protocol::gopher, protocol::http, protocol::imap,
 protocol::ipv6, protocol::kerberos, protocol::ldap, protocol::pop3,
 protocol::sftp, protocol::smtp, protocol::ssh, protocol::ssl,
 protocol::telnet, protocol::tftp, role::program, scope::utility,
 use::downloading, use::synchronizing, use::transmission,
 works-with::file, works-with::mail
```

```
Download-Size: 315 kB
APT-Sources: http://deb.debian.org/debian bookworm/main amd64 Packages
Description: Befehlszeilenwerkzeug zur Dateiübertragung mit URL-Syntax
 curl ist ein Befehlszeilenwerkzeug zur Dateiübertragung mit URL-Syntax. Es
 unterstützt DICT, FILE, FTP, FTPS, GOPHER, HTTP, HTTPS, IMAP, IMAPS, LDAP,
 LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMTP, SMTPS, TELNET und TFTP.
 .
 curl unterstützt SSL-Zertifikate, HTTP POST, HTTP PUT, Hochladen zu FTP,
 Hochladen von HTTP-Formularen, Proxys, Cookies, Benutzerauthentifizierung
 mit Name und Passwort (einfach, verschlüsselt (Digest), NTLM, Negotiate,
 Kerberos), Wiederaufnahme von Dateiübertragungen, tunneln von HTTP-Proxys
 und jede Menge weitere nützliche Tricks.
```

Paket-Inhalte (enthaltene Dateien) anzeigen:

dpkg -L

Mit diesem Kommando werden die Dateien des angegebenen Pakets (hier `curl`) aufgelistet.

```bash
$ dpkg -L curl
/.
/usr
/usr/bin
/usr/bin/curl
/usr/share
/usr/share/doc
/usr/share/doc/curl
/usr/share/doc/curl/changelog.Debian.gz
/usr/share/doc/curl/changelog.gz
/usr/share/doc/curl/copyright
/usr/share/man
/usr/share/man/man1
/usr/share/man/man1/curl.1.gz
/usr/share/zsh
```

Paket-Suche: Überblick

- `dpkg -S <file>`: Sucht nach dem installierten Paket, das die angegebene Datei enthält.
- `apt search <pattern>`: Sucht nach Paketen, die dem Paket-Namensmuster entsprechen.
- `apt-file search <pattern>`: Sucht nach Dateien in den Paketen, die dem Datei-Namensmuster entsprechen.

Installierte Pakete durchsuchen nach Dateinamen: `dpkg -S`

Das Kommando `dpkg -S` sucht nach dem installierten Paket, das die angegebene Datei enthält. Dabei kann die Datei auch mit einem Pfad angegeben werden.

```
$ dpkg -S ifconfig # assuming pkg net-tools is installed
net-tools: /sbin/ifconfig
net-tools: /usr/share/man/de/man8/ifconfig.8.gz
net-tools: /usr/share/man/man8/ifconfig.8.gz
net-tools: /usr/share/man/pt_BR/man8/ifconfig.8.gz
net-tools: /usr/share/man/fr/man8/ifconfig.8.gz
```

```
$ dpkg -S /sbin/ifconfig
net-tools: /sbin/ifconfig
```

```
$ dpkg -S /usr/bin/curl
curl: /usr/bin/curl
```

Gibt man nur den Dateinamen als Muster an, sucht `dpkg -S` nach allen Paketen, die das angegebene Muster enthalten. Wahrscheinlich erhält man dann viel mehr Treffer und die Trefferliste wird unübersichtlicher. Mit dem vollständigen Pfad wird die Suche viel präziser. Dazu muss man natürlich den genauen Pfad der Datei kennen.

```
$ dpkg -S curl
libcurl3-gnutls:amd64: /usr/share/doc/libcurl3-gnutls/changelog.Debian.gz
libcurl4:amd64: /usr/share/doc/libcurl4
curl: /usr/bin/curl
curl: /usr/share/doc/curl
libcurl3-gnutls:amd64: /usr/share/doc/libcurl3-gnutls/copyright
libcurl4:amd64: /usr/share/doc/libcurl4/changelog.gz
libcurl4:amd64: /usr/share/doc/libcurl4/changelog.Debian.gz
libcurl3-gnutls:amd64: /usr/lib/x86_64-linux-gnu/libcurl-gnutls.so.4.8.0
curl: /usr/share/doc/curl/copyright
curl: /usr/share/zsh/vendor-completions/_curl
curl: /usr/share/doc/curl/changelog.Debian.gz
libcurl4:amd64: /usr/lib/x86_64-linux-gnu/libcurl.so.4.8.0
libcurl4:amd64: /usr/lib/x86_64-linux-gnu/libcurl.so.4
libcurl3-gnutls:amd64: /usr/share/doc/libcurl3-gnutls/changelog.gz
libcurl3-gnutls:amd64: /usr/share/lintian/overrides/libcurl3-gnutls
libcurl3-gnutls:amd64: /usr/share/doc/libcurl3-gnutls
curl: /usr/share/doc/curl/changelog.gz
libcurl3-gnutls:amd64: /usr/lib/x86_64-linux-gnu/libcurl-gnutls.so.4
curl: /usr/share/man/man1/curl.1.gz
libcurl3-gnutls:amd64: /usr/lib/x86_64-linux-gnu/libcurl-gnutls.so.3
libcurl4:amd64: /usr/share/doc/libcurl4/copyright
```

Pakete durchsuchen nach Suchbegriff: `apt search`

Mit diesem Kommando wird nach Paketen gesucht, die dem angegebenen Namensmuster entsprechen. Das Kommando kann nicht nur nach Paketnamen, sondern auch nach Beschreibungen und anderen Metadaten suchen. Die Suche nach Dateien, die in den Paketen enthalten sind, ist mit `apt search` nicht möglich. Möglicherweise erhält man eine große Trefferliste.

```
$ apt search curl
```

Erweiterte Paket-Suche mit `apt-file`

`apt-file` ist ein zusätzliches Werkzeug, das den Inhalt (auch der nicht installierten) der Pakete kennt und auch über die enthaltenen Dateien Auskunft geben kann. `apt-file` muss zuerst installiert und dann initialisiert werden, d.h. die Datei-Datenbank muss zuerst aus dem Internet heruntergeladen und lokal gespeichert werden.

```
$ sudo apt install apt-file # install it
...
$ sudo apt-file update # update the file database
$ # now you can search for files in packages; apt-file queries the local db
```

```
$ apt-file search curl # retrieves a long list of matches
...
```

Das Suchmuster 'curl' ist recht unspezifisch und liefert eine lange Liste von Treffern. Je genauer das Suchmuster, desto präziser die Trefferliste. Die Angabe des absoluten Pfades der Datei ist häufig eine gute Wahl und liefert eine kleine Trefferliste.

```
$ apt-file search /usr/bin/curl
curl: /usr/bin/curl
libcurl4-gnutls-dev: /usr/bin/curl-config
libcurl4-nss-dev: /usr/bin/curl-config
libcurl4-openssl-dev: /usr/bin/curl-config
libcurlpp-dev: /usr/bin/curlpp-config
```

Beispielsuche nach `add-apt-repository`

Zum Anzeigen, Hinzufügen und Entfernen von Paket-Quellen wird das Kommando `add-apt-repository` verwendet. Dieses Kommando ist nicht Bestandteil der Standard-Installation von Debian. Die Frage ist: Welches Paket enthält das Kommando `add-apt-repository`, damit es installiert werden kann und wir dann das Kommando verwenden können?

- `dpkg -S add-apt-repository` liefert keine Treffer, weil das Kommando nicht installiert ist.

- `apt search add-apt-repository` liefert keine Treffer, weil das Suchmuster `add-apt-repository` nicht im Paketnamen oder in den Paket-Metadaten vorkommt.
- `apt-file search add-apt-repository` liefert den Treffer, weil `apt-file` den Inhalt der (nicht installierten) Pakete kennt und auch über die enthaltenen Dateien Auskunft geben kann.

```
$ apt-file search add-apt-repository
puppet-module-puppetlabs-apt: /usr/share/puppet/.../add-apt-repository.sh.epp
software-properties-common: /usr/bin/add-apt-repository
software-properties-common: /usr/share/bash-completion/completions/add-apt-repository
software-properties-common: /usr/share/man/man1/add-apt-repository.1.gz
```

Auch ohne absoluten Pfad findet `apt-file` nur zwei Pakete, in denen das Mustern `add-apt-repository` vorkommt. Wir sehen, dass das Kommando `/usr/bin/add-apt-repository` im Paket `software-properties-common` enthalten ist. Also muss das Paket `software-properties-common` installiert werden, um das Kommando `add-apt-repository` verwenden zu können.

```
$ sudo apt install software-properties-common
```

Paket-Abhängigkeiten

Pakete können voneinander abhängen. Ein Paket kann andere Pakete benötigen, um korrekt zu funktionieren. Diese Abhängigkeiten sind in den Metadaten der Pakete festgelegt. `apt` kann die Abhängigkeiten auflösen und die benötigten Pakete automatisch mitinstallieren.

Ist Paket A von Paket B abhängig und Paket B von Paket C, spricht man von einer **transitiven Abhängigkeit**: A ist transitiv abhängig von C.

So entsteht ein ganzer Baum von Abhängigkeiten.

Abhängigkeiten anzeigen: `apt depends <package>`

```
$ apt depends tar
ar
  Hängt ab von (vorher): libacl1 (>= 2.2.23)
  Hängt ab von (vorher): libc6 (>= 2.34)
  Hängt ab von (vorher): libselinux1 (>= 3.1~)
  Kollidiert mit: cpio (<= 2.4.2-38)
  Beschädigt: dpkg-dev (<< 1.14.26)
  Schlägt vor: bzip2
  Schlägt vor: ncompress
  Schlägt vor: xz-utils
  Schlägt vor: tar-scripts
  Schlägt vor: <tar-doc>
  Ersetzt: cpio (<< 2.4.2-39)
```

- `apt depends` zeigt die direkten Abhängigkeiten des angegebenen Pakets (hier `tar`) an.
- `apt-rdepends` zeigt die recursiven Abhängigkeiten des angegebenen Pakets an.
- `apt-rdepends` ist nicht Bestandteil der Standard-Installation von Debian. Es muss zuerst installiert werden.

Rekursive Abhängigkeiten anzeigen: `apt-rdepends <package>`

```
$ sudo apt install apt-rdepends
```

```
$ apt-rdepends tar
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
tar
  PreDepends: libacl1 (>= 2.2.23)
  PreDepends: libc6 (>= 2.34)
  PreDepends: libselinux1 (>= 3.1~)
libacl1
  Depends: libc6 (>= 2.33)
libc6
  Depends: libgcc-s1
libgcc-s1
  Depends: gcc-12-base (= 12.2.0-14)
  Depends: libc6 (>= 2.35)
gcc-12-base
libselinux1
  Depends: libc6 (>= 2.34)
  Depends: libpcre2-8-0 (>= 10.22)
libpcre2-8-0
  Depends: libc6 (>= 2.34)
```

Paket-Quellen

`sources.list` und `sources.list.d/*.list`

In Debian-Linux-Systemen und Derivaten sind die Paket-Quellen in der Datei `/etc/apt/sources.list` und in den Dateien `/etc/apt/sources.list.d/*.list` konfiguriert. Die Datei `sources.list` enthält die Standard-Paket-Quellen. Die Dateien im Verzeichnis `sources.list.d` enthalten zusätzliche Paket-Quellen.

Jede gültige Zeile (die nicht leer ist und nicht mit `#` beginnt (Kommentarzeile)) in diesen Dateien wird als Paket-Quelle interpretiert.

```
$ # show the content of the sources.list and the files in sources.list.d w/o comments
hermann@deb-srv:~$ cat /etc/apt/sources.list /etc/apt/sources.list.d/*.list 2>/dev/null |
> grep -v '^ *#' | grep -v '^$'
deb http://deb.debian.org/debian/ bookworm main non-free-firmware
deb-src http://deb.debian.org/debian/ bookworm main non-free-firmware
deb http://security.debian.org/debian-security bookworm-security main non-free-firmware
deb-src http://security.debian.org/debian-security bookworm-security main non-free-firmware
deb http://deb.debian.org/debian/ bookworm-updates main non-free-firmware
deb-src http://deb.debian.org/debian/ bookworm-updates main non-free-firmware
```

Diese Ausgabe zeigt die Standard-Paket-Quellen für das Debian-System (Leerzeilen und Kommentarzeilen mit `grep` entfernt).

Jeder Eintrag genügt einem bestimmten Format:

- `deb` **oder** `deb-src`: gibt an, ob es sich um eine Binär- oder Quellpaket-Quelle handelt.
- **URL**: die URL der Paket-Quelle.
- **Distribution**: die Codename der Distribution (hier `bookworm`, etc.).
- **1. Komponente**: die Komponenten der Distribution (hier `main`)
- **Optionale Komponenten**: weitere Komponenten der Distribution (hier `non-free-firmware`)

Hinzufügen und Entfernen von Paket-Quellen

Früher war es üblich, die Paket-Quellen in den Paket-Quellen-Dateien direkt mit einem Editor (`nano`, `vim`, etc.) zu bearbeiten. Heute nimmt man dazu das Kommando `add-apt-repository`, das auch die Syntax der Paket-Quellen-Einträge prüft und Fehlermeldungen ausgibt, wenn die Syntax nicht korrekt ist.

Das Kommando `add-apt-repository` (aus dem Paket `software-properties-common`) kann verwendet werden, um Paket-Quellen anzuzeigen, hinzuzufügen und zu entfernen.

Mit dem Hinzufügen von Paket-Quellen ist Vorsicht geboten. Die Paket-Quellen sollten nur vertrauenswürdige Quellen sein, um die Sicherheit des Systems nicht zu gefährden. Zur Überprüfung der Quellen kann das Kommando `apt-key` verwendet werden, um die GPG-Schlüssel der Quellen zu überprüfen. Auch diese Schlüssel müssen ggf. zunächst heruntergeladen und in das `apt` -System importiert werden. Mehr dazu unter diesen Links:

- https://linuxize.com/post/how-to-add-apt-repository-in-ubuntu/
- https://phoenixnap.de/Glossar/persönliches-Paketarchiv

Vor- und Nachteile zusätzlicher Paket-Quellen

Vorteile:

- Zusätzliche Software, die nicht in den Standard-Paket-Quellen enthalten ist, kann installiert werden.
- Verfügbare neuere Versionen von Software, die in den Standard-Paket-Quellen nicht so schnell aktualisiert werden.

Nachteile:

- Sicherheitsrisiko: Die Quellen könnten nicht vertrauenswürdig sein und Schadsoftware enthalten.
- Stabilität: Die zusätzlichen Paket-Quellen könnten instabile oder ungetestete Software enthalten, die das System instabil machen.
- Konflikte: Die zusätzlichen Paket-Quellen könnten Pakete enthalten, die in Konflikt mit den Standard-Paket-Quellen stehen. Beispielsweise werden von den zusätzlichen Quellen Bibliotheken in anderen Versionen bereitgestellt als diejenigen, die von den Standard-Paket-Quellen bereitgestellt werden.

Empfehlungen zu zusätzlichen Paket-Quellen

Die Standard-Paket-Quellen enthalten eine große Auswahl an Software und sind in der Regel ausreichend für die meisten Anwendungsfälle. Außerdem sind die Pakete in den Standard-Paket-Quellen gut kuratiert, getestet und stabil. Man erhält regelmäßige Sicherheits-Updates und Bugfixes. Allerdings ist bis zum nächsten Release der Distribution die Software nicht immer auf dem neuesten Stand.

Es ist ratsam, zusätzliche Paket-Quellen nur dann hinzuzufügen, wenn es wirklich notwendig ist und die Quellen vertrauenswürdig sind.

Diese Entscheidung ist auch mit Blick auf die Art der Verwendung eines Linux-Systems zu treffen:

- Server oder Desktop
- Produktiv-, Test- oder Entwicklungssystem
- Betriebliche oder private Nutzung.

Server- vs. Desktop-Systeme

- **Server-Systeme**: dort wird in der Regel nur die Software installiert, die für den Server-Betrieb notwendig ist. Dies kann meist mit den Standard-Paket-Quellen abgedeckt werden. Doch auch hier kann es vorkommen, dass zusätzliche Software benötigt wird, die nicht in den Standard-Paket-Quellen enthalten ist. Dies ist bei Datenbank-Servern häufig der Fall (z.B. MariaDB, MongoDB, Cassandra, OracleDB, etc.). Hier kann es jedoch sinnvoll sein, die Datenbank-Software in einem Docker-Container zu betreiben.

- **Desktop-Systeme**: hier wird oft zusätzliche Software installiert, die nicht in den Standard-Paket-Quellen enthalten ist. Es kann sinnvoll sein, zusätzliche Paket-Quellen hinzuzufügen. Stattdessen kann auch auf die Pakete der Software-Verwaltungen von Flatpak oder Snap (nur Ubuntu) zurückgegriffen werden.

Produktiv-, Test- oder Entwicklungssysteme

- **Produktivsysteme**: dulden keine Kompromisse bei Stabilität und Sicherheit. Von den Standard-Paket-Quellen wird nur abgewichen, wenn es sich um namhafte Hersteller handelt.
- **Testsysteme**: sollen in der Regel die Produktivsysteme möglichst genau abbilden. Deshalb gelten hier die gleichen Regeln wie für Produktivsysteme.
- **Entwicklungssysteme**: erlauben eine größere Freiheit bei der Software-Auswahl. Hier kann es sinnvoll sein, zusätzliche Paket-Quellen hinzuzufügen, um die neuesten Versionen von Software zu testen.

Betriebliche oder private Nutzung

- **Betriebliche Nutzung**: erfordert eine größere Sorgfalt bei der Auswahl der Software. Häufig wird hier nur Software aus den Standard-Paket-Quellen installiert. Oft ist auch dies noch eingeschränkt. Normale Benutzer können häufig keine Software installieren.
- **Private Nutzung**: erlaubt eine größere Freiheit bei der Software-Auswahl. Der Privatanwender ist auch selbst für die Stabilität und Sicherheit seines Systems verantwortlich. Will der Privatnutzer Anwendungen wie Spiele oder Multimedia-Software installieren, kann es dennoch sinnvoll sein, auf die Pakete von Flatpak oder Snap zurückzugreifen.