

Einführung in das Paketmanagementsystem Portage

Erste Schritte im Paketmanagement der Gentoo-Distribution haben Sie bereits vollzogen – durch die Installation von Kernel-Quellen, einem System Logger und weiteren essenziellen Programmen zum Betrieb eines Gentoo-Systems.

Dieses Kapitel bietet eine Einführung in die wichtigsten Funktionen des Paketmanagementsystems Portage, die fortwährende Aktualisierung des Systems sowie die Installation und Deinstallation von Programmen.

Nach diesem Kapitel können Sie die wichtigsten Aufgaben zur Pflege und Anpassung eines Gentoo-Systems durchführen.

3.1 Das Frontend »emerge«

Das Kommandozeilen-Frontend `emerge` vereint einerseits den in der Skriptsprache Python programmierten Teil des Paketmanagementsystems und ist andererseits das Frontend für den in Shellskripten implementierten Teil des Paketmanagements.

Frontend? Ja, theoretisch können Sie die Installation eines Pakets auch mit `ebuild` durchführen. In einem frühen Entwicklungsstadium des Paketmanagements war dies gar der einzige Weg zur Installation eines Pakets.

Grundsätzlich möglich ist diese Art der Installation auch heute noch. Genau genommen übernimmt das Frontend `emerge` nur diese Funktion(en), löst aber darüber hinaus die in Paketen bestimmten Abhängigkeiten auf und ermöglicht die Installation der »besten« Version eines Pakets.

Hinweis

Neben dem Kommandozeilen-Frontend `emerge` gibt es zahlreiche weitere grafische Frontends, wie beispielsweise `kuroo` oder `porthole`. Diese werden jedoch im Gegensatz zu `emerge` nicht von den Entwicklern als Teil des Paketmanagementsystems entwickelt und gepflegt. Daher ist eine vollständige Funktion dieser Programme nicht immer garantiert. Dies wird sich jedoch voraussichtlich mit Festlegung der EAPI ändern, in der Funktionalität und Verhalten des Paketmanagementsystems festgeschrieben werden.

3.1.1 ebuild

`ebuild` ist eine direkte Schnittstelle zum Paketmanagement, mit der Kommandos direkt auf der Ebene eines einzelnen Ebuilds ausgeführt werden. Die Syntax verlangt immer den Pfad zu einem Ebuild sowie ein oder mehrere auszuführende Kommandos. Liegt das Ebuild nicht im aktuellen Arbeitsverzeichnis, ist es notwendig, den vollständigen Pfad anzugeben.

Gültige Kommandos unterstützen unter anderem das Entpacken der benötigten Quell-Archive (`unpack`), das Konfigurieren und Kompilieren der Quellen (`compile`), das Installieren des Programms (`install`) und die eigentliche Installation in das System (`qmerge`).

Diese Kommandofolge wird wiederum durch das Kommando `merge` zusammengefasst, so dass `ebuild hdparm-6.6.ebuild merge` generell mit `emerge hdparm` gleichzusetzen ist – bis auf die beschriebenen Unterschiede in der Auflösung von Abhängigkeiten und der Installation einer fix vorgegebenen Version einerseits und der Installation der besten verfügbaren Version andererseits.

```

hammer hdparm # ebuild hdparm-6.6.ebuild merge
Disabling noauto in features... merge disables it. (qmerge doesn't)
* hdparm-6.6.tar.gz RMD160 (-) ... [ ok
* hdparm-6.6.tar.gz SHA1 (-) ... [ ok
* hdparm-6.6.tar.gz SHA256 (-) ... [ ok
* hdparm-6.6.tar.gz size (-) ... [ ok
* checking ebuild checksums (-) ... [ ok
* checking auxfile checksums (-) ... [ ok
* checking miscfile checksums (-) ... [ ok
* checking hdparm-6.6.tar.gz (-) ... [ ok
>>> Unpacking source...
>>> Unpacking hdparm-6.6.tar.gz to /var/tmp/portage/sys-apps/hdparm-6.6/work
>>> Source unpacked.
>>> Compiling source in /var/tmp/portage/sys-apps/hdparm-6.6/work/hdparm-6.6 ...
i686-pc-linux-gnu-gcc -O2 -march=prescott -pipe -W -Wall -Wbad-function-cast -Wcast-align -Wpointer-arith -Wcast-qual -Wshadow -Wstrict-prototypes -Wmissing-prototypes -Wmissing-declarations -fkeep-inline-functions -Wwrite-strings -Waggregate-return -Wnested-externs -Wtrigraphs -O2 -march=prescott -pipe -c -o hdparm.o hdparm.c
i686-pc-linux-gnu-gcc -O2 -march=prescott -pipe -W -Wall -Wbad-function-cast -Wcast-align -Wpointer-arith -Wcast-qual -Wshadow -Wstrict-prototypes -Wmissing-prototypes -Wmissing-declarations -fkeep-inline-functions -Wwrite-strings -Waggregate-return -Wnested-externs -Wtrigraphs -O2 -march=prescott -pipe -c -o identify.o identify.c
hdparm.c: In function 'process_dev':
hdparm.c:1991: warning: 'packed' attribute ignored for field of type 'char[512]'
i686-pc-linux-gnu-gcc -o hdparm hdparm.o identify.o
>>> Source compiled.
>>> Test phase [none]: sys-apps/hdparm-6.6
>>>
>>> Install hdparm-6.6 into /var/tmp/portage/sys-apps/hdparm-6.6/image/ category sys-apps
>>> Completed installing hdparm-6.6 into /var/tmp/portage/sys-apps/hdparm-6.6/image/
decompressdir: bzip2 -9 /usr/share/man/man8
strip: i686-pc-linux-gnu-strip --strip-unneeded
  sbin/hdparm
>>> Merging sys-apps/hdparm-6.6 to /
  * Removing /usr/share/info
  * Removing /usr/share/doc
  --- /usr/
  --- /usr/share/
  --- /usr/share/man/
  --- /usr/share/man/man8/
>>> /usr/share/man/man8/hdparm.8.bz2
  --- /etc/
  --- /etc/conf.d/
>>> /etc/conf.d/hdparm
  --- /etc/init.d/
>>> /etc/init.d/hdparm
  --- /sbin/
>>> /sbin/hdparm
>>> /sbin/identl
>>> sys-apps/hdparm-6.6 merged.

```

Abb. 3.1: `ebuild hdparm-6.6.ebuild merge`

Der direkte Zugriff auf das Paketmanagementsystem mittels `ebuild` ist zwar weiterhin möglich, aber nicht zwingend erforderlich. Das Frontend `emerge` beinhaltet die Funktionalität von `ebuild` und bietet essenzielle, zusätzliche Funktionalitäten.

Eine potenzielle Ausnahme stellt das Kommando `digest` dar, das die zu einem Ebuild benötigten Prüfsummen erstellt – dies ist zumeist jedoch nur bei der Entwicklung eigener Ebuilds erforderlich.

3.1.2 Aktionen und Optionen des Paketmanagers

Die folgenden Tabellen listen Aktionen und Optionen des Paketmanagers auf, inklusive der – soweit vorhandenen – Kurzform und einer Beschreibung.

Die Übergabe der vom Paketmanager zu nutzenden Aktionen und Optionen hat sich in letzten Versionen marginal verändert, während zuvor die Form `emerge aktion` üblich war, wird nun `emerge --aktion` genutzt. In den meisten Fällen ist die ältere Notation noch verfügbar, Sie werden jedoch darauf hingewiesen, dass die Notation veraltet ist.

Funktion	Kurzform	Beschreibung
<code>--clean</code>		Entfernt alle Versionen eines Pakets, mit Ausnahme der aktuellsten Version.
<code>--config</code>		Führt die Konfiguration eines installierten Pakets aus.
<code>--depclean</code>		Entfernt verwaiste Pakete, die weder explizit installiert wurden noch als Abhängigkeit eines anderen Pakets benötigt werden.
<code>--help</code>		Zeigt die integrierte Hilfe an.
<code>--info</code>		Zeigt Informationen über die Gentoo-Installation an.
<code>--metadata</code>		Kopiert den Metadaten-Cache, dies geschieht nach einem <code>--sync</code> automatisch.
<code>--prune</code>	<code>-P</code>	Wie <code>--clean</code> , beachtet jedoch nicht unterschiedliche SLOTS.
<code>--regen</code>		Aktualisiert den Cache, anhand dessen Portage Abhängigkeiten auflöst.
<code>--resume</code>		Wiederholt die letzte Installationsoperation.
<code>--search</code>	<code>-s</code>	Durchsucht den Portage Tree nach Paketnamen.
<code>--searchdesc</code>	<code>-S</code>	Durchsucht den Portage Tree, zieht aber auch Paketbeschreibungen in die Suche mit ein.
<code>--sync</code>		Aktualisiert den Portage Tree.
<code>--unmerge</code>	<code>-C</code>	Deinstalliert ein Paket.
<code>--update</code>	<code>-u</code>	Aktualisiert ein Paket oder ein Target (<code>system world</code>).
<code>--version</code>	<code>-v</code>	Gibt die Version des Paketmanagers aus.

Tabelle 3.1: Portage-Aktionen

Die in der folgenden Tabelle gelisteten Optionen können im Gegensatz zu den Aktionen kombiniert werden, beispielsweise `emerge --verbose --pretend`.

Option	Kurzform	Beschreibung
<code>--ask</code>	<code>-a</code>	Fragt nach <code>--pretend</code> -Ausgaben, ob die gewünschte Operation ausgeführt werden soll.
<code>--buildpkg</code>	<code>-b</code>	Baut Binärpakete und installiert diese.
<code>--buildpkgonly</code>	<code>-B</code>	Baut ausschließlich Binärpakete, installiert diese aber nicht.
<code>--changelog</code>	<code>-l</code>	Zeigt den Change-Log zu einem Paket an.
<code>--deep</code>	<code>-D</code>	Bezieht den vollständigen Abhängigkeitspfad mit ein, anstelle nur der minimal benötigten Abhängigkeiten.
<code>--emptytree</code>	<code>-e</code>	Stößt eine Reinstallation von allen installierten Paketen an.
<code>--fetchonly</code>	<code>-f</code>	Lädt nur die Quell-Archive, nimmt aber keine Installation vor.
<code>--ignore-default-opts</code>		Übergeht die in der Variablen <code>EMERGE_DEFAULT_OPTS</code> vorgegebenen Standardoptionen.
<code>--newuse</code>	<code>-N</code>	Prüft, ob bei Paketen eine geänderte USE-Flag-Konfiguration vorhanden ist.
<code>--nodeps</code>	<code>-O</code>	Installiert nur die angegebenen Programme, nicht deren Abhängigkeiten.
<code>--noreplace</code>	<code>-n</code>	Verhindert die erneute Installation von Programmen, die bereits installiert wurden.
<code>--oneshot</code>	<code>-1</code>	Installiert ein Paket, trägt dieses aber nicht in die World-Datei ein.
<code>--onlydeps</code>	<code>-o</code>	Installiert nur die Abhängigkeiten eines Pakets.
<code>--pretend</code>	<code>-p</code>	Führt das angegebene Kommando in einem »Was wäre wenn«-Modus aus und zeigt die zu unternehmenden Schritte an.
<code>--quiet</code>	<code>-q</code>	Erzeugt weniger Ausgaben, die Menge verbleibender Ausgaben hängt von der gewählten Aktion ab.
<code>--skipfirst</code>		Übergeht das erste Paket in der Liste zu installierender Pakete.
<code>--tree</code>	<code>-t</code>	Zeigt den vollständigen Abhängigkeitsbaum an.
<code>--usepkg</code>	<code>-k</code>	Benutzt, wenn vorhanden, Binärpakete anstelle der Installation aus Quell-Archiven.

Tabelle 3.2: Portage-Optionen

Option	Kurzform	Beschreibung
<code>--usepkgonly</code>	<code>-K</code>	Benutzt ausschließlich Binärpakete, sind diese nicht vorhanden, schlägt die Aktion fehl.
<code>--verbose</code>	<code>-v</code>	Verleiht Portage die Gesprächigkeit, das Maß der Gesprächigkeit variiert von Aktion zu Aktion. Sinnvoll bei <code>-pretend</code> .

Tabelle 3.2: Portage-Optionen (Forts.)

3.2 Wo konfiguriere ich Portage?

Die Konfiguration des Paketmanagementsystems gliedert sich in zwei Bereiche:

- Globale, allgemeingültige Konfiguration
- Konfiguration pro Paket

Mit der Veröffentlichung der Version 2.0.50 des Paketmanagementsystems Portage im Herbst 2004, und späteren weiteren Versionen, wurde die Möglichkeit geschaffen und erweitert, die Konfiguration auf Ebene eines einzelnen Pakets durchzuführen. Dadurch wird vor allem die Konfiguration von Ausnahmen stark vereinfacht, so dass es seitdem problemlos möglich ist, in einem stabilen System einzelne Pakete aus dem Test-Zweig der Distribution zu nutzen – oder global für alle Programme die Nutzung einer optionalen Komponente über die Deaktivierung des USE-Flags zu verhindern, mit Ausnahme eines Pakets, in dem diese optionale Komponente benötigt wird.

Die globale, allgemeingültige Konfiguration erfolgt über die Datei `/etc/make.conf`, die lokale Konfiguration pro Paket über unterschiedliche Dateien im Verzeichnis `/etc/portage/`.

3.2.1 Konfigurationsdateien im Überblick

Um einen besseren Überblick über die verschiedenen von Portage genutzten Konfigurationsdateien und Verzeichnisse zu erlangen, sind diese hier kurz dargestellt.

Datei/Verzeichnis	Funktion/Zweck
<code>/etc/make.conf</code>	Globale Konfiguration
<code>/etc/portage/</code>	Lokale, paketbezogene Konfiguration Anpassung des Paketmanagements
<code>/etc/portage/package.use</code>	Paketbezogene Konfiguration von USE-Flags
<code>/etc/portage/package.keywords</code>	Paketbezogene Konfiguration des zu nutzenden Zweiges

Tabelle 3.3: Konfigurationsdateien und Verzeichnisse

Datei/Verzeichnis	Funktion/Zweck
/etc/portage/package.mask	Paketbezogene Maskierung von Paketen oder einzelnen Versionen von Paketen
/etc/portage/package.unmask	Paketbezogene Demaskierung von Paketen oder einzelnen Versionen von Paketen
/var/lib/portage/	Enthält die World-Datei und die Prüfsummen der Konfigurationsdateien
/var/lib/portage/world	Enthält alle direkt installierten Pakete, d.h. ohne Abhängigkeiten.
/var/lib/portage/config	Enthält Prüfsummen von Konfigurationsdateien, siehe Abschnitt 6.3
/var/db/	Enthält Paketdatenbanken.
/var/db/pkg/	Enthält die Portage-Paketdatenbank.
/var/db/webapps/	Enthält die Paketdatenbank von mit <code>webapp-config</code> installierten Webapplikationen.
/var/cache/edb/	Enthält unter anderem den Metadaten-Cache zur Beschleunigung von Suchvorgängen und der Auflösung von Abhängigkeiten.

Tabelle 3.3: Konfigurationsdateien und Verzeichnisse (Forts.)

3.3 Die Atomisierung des Portage Tree

In Kapitel 2, vor der eigentlichen Installation, stand die Erklärung der Begrifflichkeiten »Stage-Archiv« und Portage Tree. Während sich ein Stage-Archiv nahezu ausschließlich auf die Installation bezieht, beinhaltet der Portage Tree alle Daten des Paketmanagementsystems. Bevor also ein tieferer Einblick in das Paketmanagementsystem stattfinden kann, ist eine Atomisierung, ein Zerlegen in die kleinsten Teile, für das weitere Verständnis erforderlich.

3.3.1 Ein »Paket«?

Zunächst eine vielleicht unwichtig erscheinende Betrachtung – wir sprechen von einem Paketmanagementsystem und einzelnen Paketen auf der einen Seite und Anpassbarkeit und Individualität einer Metadistribution auf der anderen Seite.

Unter der Beschreibung »Paket« findet sich zwangsläufig die Assoziation zu einem Programm und dessen elementaren Dateien. In einem weiteren Schritt assoziieren wir mit einem »Paket« die Installation eines Programms in einer binären Distribution. Das unter anderem von SuSE, Red Hat und Mandriva genutzte RPM-Format trägt das »Paket« nicht umsonst bereits im Namen und Debian-Archive werden

weitläufig auch als »Debian-Paket« bezeichnet. Worin unterscheidet sich die Metadistribution Gentoo also von anderen binären Distributionen – wenn doch bei beiden »Pakete« zur Installation kommen?

Schauen wir uns daher als Beispiel ein Debian-Paket ein wenig genauer an:

```
debian:~# ar t /var/cache/apt/archives/file_4.17-5etch1_i386.deb
debian-binary
control.tar.gz
data.tar.gz
```

Ein Debian-Paket besteht aus einer Datei `debian-binary`, in der die Version des Debian-Paket-Formats beschrieben ist, und zwei Tar-Archiven.

`control.tar.gz` beinhaltet Metadaten in Form von Prüfsummen der einzelnen Bestandteile des `data.tar.gz`-Archivs und einer Beschreibung, zu der Informationen über die Kategorie, Architektur, Abhängigkeiten und den Paketbetreuer gehören.

```
Package: file
Version: 4.17-5etch1
Section: utils
Priority: standard
Architecture: i386
Depends: libmagic1 (= 4.17-5etch1), libc6 (>= 2.3.6-6), libmagic1
Installed-Size: 116
Maintainer: Michael Piefel piefel@debian.org
Description: Determines file type using "magic" numbers
 File tests each argument in an attempt to classify it. There are three
 sets of tests, performed in this order: filesystem tests, magic number
 tests, and language tests. The first test that succeeds causes the
 file type to be printed.
.
 Starting with version 4, the file command is not much more than a wrapper
 around the "magic" library.
```

Listing 3.1: Metadaten innerhalb eines Debian-Pakets

`data.tar.gz` beinhaltet die eigentliche Anwendung sowie dazugehörige Konfigurationsdateien und Dokumentation.

```
./usr
./usr/bin
./usr/bin/file
./usr/share
./usr/share/man
./usr/share/man/man1
./usr/share/man/man1/file.1.gz
./usr/share/lintian
./usr/share/lintian/overrides
./usr/share/lintian/overrides/file
./usr/share/doc
./usr/share/doc/file
./usr/share/doc/file/changelog.Debian.gz
./usr/share/doc/file/copyright
./usr/share/doc/file/changelog.gz
./usr/share/bug
./usr/share/bug/file
./usr/share/bug/file/presubj
./etc
./etc/magic
```

Listing 3.2: Inhalte des `data.tar.gz` innerhalb eines Debian-Pakets

Lassen wir nun die Version der das Debian-Paketformat beschreibende `debian-binary`-Datei außen vor, so bleiben im nächsten Schritt zwei Bestandteile innerhalb eines Debian-Pakets übrig:

- Metadaten und Prüfsummen
- die eigentliche Anwendung oder Systembibliothek(en)

Diese beiden Bestandteile eines »Pakets« lassen sich nun auch auf Struktur und Aufbau der Gentoo-Distribution abbilden. Ein »Paket« besteht auch hier aus zwei Bestandteilen, den Metadaten und Prüfsummen und der eigentlichen Anwendung bzw. Systembibliothek.

Der Unterschied ist jedoch die Form der Distribution – anstelle eines Archivs mit allen Bestandteilen verbreitet Gentoo innerhalb seines Portage Tree Metadaten und Prüfsummen, das eigentliche »Paket« entsteht erst durch Kompilierung unter Zuziehung dieser Metadaten durch den Anwender.

Wird im Folgenden also von einem »Paket« gesprochen, so bezieht sich das Paket auf den innerhalb des Portage Tree verbreiteten Bestandteil – Ebuild, Metadaten, Prüfsummen usw.

Hinweis

Die Betrachtung eines »Pakets« lässt sich recht ähnlich auch auf RPM-basierte Distributionen übertragen. Das RPM-Format beinhaltet ebenfalls unterschiedliche Bestandteile – Metadaten und das eigentliche Programm – in einem »Paket«.

3.3.2 Schritt 1: Portage Tree

Der Begriff »Portage Tree« umfasst alle Pakete innerhalb der Gentoo-Distribution. Sofern nicht explizit anders konfiguriert, befindet sich die lokale Kopie des Portage Tree im Verzeichnis `/usr/portage`.

In einem ersten Schritt sind diese zur besseren Übersichtlichkeit in thematische Kategorien aufgeteilt. Über diese Kategorien hinaus befinden sich weitere Verzeichnisse und Dateien innerhalb des Portage Tree:

- `distfiles`: In diesem Verzeichnis werden die Quellen der einzelnen Programme gespeichert.
- `eclass`: In diesem Verzeichnis werden in `eclasses` ausgelagerte Funktionen gespeichert.
- `licenses`: In diesem Verzeichnis werden alle von verbreiteten Programmen genutzten Lizenzen gespeichert.
- `metadata`: In diesem Verzeichnis werden verschiedene Metadaten gespeichert, dazu gehören ein Cache des Portage Tree zur Optimierung von Suchanfragen, DTD-Dateien zur Definition verschiedener XML-Datentypen und sämtliche veröffentlichten Gentoo Linux Security Announcements (GLSA) im XML-Format.
- `profiles`: Dieses Verzeichnis beinhaltet Profile, die für die von Gentoo unterstützten Architekturen und Veröffentlichungen Voreinstellungen für das Paketmanagementsystem setzen.
- `scripts`: Dieses Verzeichnis beinhaltet das `bootstrap.sh`-Skript zum Installieren eines Gentoo-Systems ausgehend von einem Stage-1-Archiv.
- Zusätzlich beinhaltet der Portage Tree die Datei `header.txt` sowie die Vorlagendateien (»Skeletons«) `skel.ChangeLog`, `skel.ebuild` und `skel.metadata.xml`.

```
tobias@homer /usr/portage $ ls
app-accessibility dev-db games-engines media-gfx net-www sys-cluster
app-admin dev-dotnet games-fps media-libs net-zope sys-devel
app-antivirus dev-embedded games-kids media-plugins packages sys-freebds
app-arch dev-games games-misc media-radio perl-core sys-fs
app-backup dev-haskell games-mud media-sound profiles sys-kernel
app-benchmarks dev-java games-puzzle media-tv rox-base sys-libs
app-cdr dev-lang games-roguelike media-video rox-extra sys-power
app-crypt dev-libs games-rpg metadata sci-astronomy sys-process
app-dicts dev-lisp games-server net-analyzer sci-biology virtual
app-doc dev-ml games-simulation net-dialup sci-calculators www-apache
app-editors dev-perl games-sports net-dns sci-chemistry www-apps
app-emacs dev-php games-strategy net-firewall www-client
app-emulation dev-php4 games-util net-fs sci-geosciences www-misc
app-forensics dev-php5 gnome-base net-ftp sci-libs www-servers
app-i18n dev-python gnome-extra net-im sci-mathematics x11-apps
app-laptop dev-ruby gnustep-apps net-irc sci-misc x11-base
app-misc dev-scheme gnustep-base net-libs sci-physics x11-drivers
app-mobilephone dev-tcltk gnustep-libs net-mail sci-visualization x11-libs
app-office dev-tex header.txt net-misc net-misc scripts x11-misc
app-pda dev-tinyos kde-base net-nds sec-policy x11-plugins
app-portage dev-util kde-misc net-news skel.ChangeLog x11-protocols
app-shells distfiles licenses net-nntp skel.ebuild x11-terms
app-text eclclass local net-p2p skel.metadata.xml x11-themes
app-vim games-action mail-client net-print sys-apps x11-wm
app-xemacs games-arcade mail-filter net-proxy sys-auth xfce-base
dev-ada games-board mail-mta net-voip sys-block xfce-extra
dev-cpp games-emulation media-fonts net-wireless sys-boot
```

Abb. 3.2: Kategorieansicht des Portage Tree

3.3.3 Schritt 2: Kategorien

Jeder durch ein Verzeichnis abgebildeten Kategorie sind einzelne Programme, besser Pakete, zugeordnet. Die Kategorie `app-portage` beispielsweise beinhaltet Anwendungen rund um das Paketmanagement, die `games-*`-Kategorien beinhalten weiter kategorisiert verschiedene Arten von Spielen und unter `gnome-base` sind die Basiskomponenten des GNOME-Desktops einsortiert.

Die Verknüpfung Paket – Kategorie und Kategorie – Paket funktioniert jedoch nicht immer eindeutig, da ein Paket, je nach Betrachtungsweise, durchaus nicht nur in einer Kategorie gut aufgehoben ist.

```
tobias@homer /usr/portage $ cd app-portage/
tobias@homer /usr/portage/app-portage $ ls
cfg-update elogviewer gatt-svn getdelta metagen portage-utils
conf-update emerge-delta-webrsync g-cpan herdstat mirrorselect porthole
deltup epm genflags klogviewer perl-info profuse
demerge esearch genlop kuroo portage-manpages splat
eix euses gentoolkit layman portagemaster udept
elogv flagedit gentoolkit-dev metadata.xml portage-mod jabber ufed
```

Abb. 3.3: Pakete innerhalb der Kategorie `app-portage`

Zusätzlich ist in jeder Kategorie eine `metadata.xml`-Datei vorhanden, in der die Kategorie kurz in mehreren Sprachen beschrieben wird.

```
tobias@homer /usr/portage/app-portage $ cat metadata.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE catmetadata SYSTEM "http://www.gentoo.org/dtd/metadata.dtd">
```

```
<catmetadata>
  <longdescription lang="en">
    The app-portage category contains software which works with
    portage or ebuilds.
  </longdescription>
  <longdescription lang="de">
    Die Kategorie app-portage enthält Programme für das Arbeiten
    mit Portage oder Ebuilds.
  </longdescription>
</catmetadata>
```

Listing 3.3: metadata.xml in der Kategorie app-portage

3.3.4 Schritt 3: Pakete

```
tobias@homer /usr/portage/app-portage $ cd portage-utils/
tobias@homer /usr/portage/app-portage/portage-utils $ ls
ChangeLog Manifest portage-utils-0.1.23.ebuild portage-utils-0.1.25.ebuild
files metadata.xml portage-utils-0.1.24.ebuild
```

Abb. 3.4: Das Paket portage-utils

Ein einzelnes Paket besteht in der Gentoo-Distribution aus mehreren Komponenten. Wichtigster Teil eines Pakets sind Ebuilds – kleine Bash-Skripte, versehen mit zusätzlichen Metadaten, die die Kompilierung und Installation eines Pakets steuern.

Zusätzlich sind jedoch für jedes Paket zwingend weitere Dateien vorhanden:

- **ChangeLog**, wie der Name beschreibt, eine Log-Datei, in der Aktivitäten und Veränderungen an dem jeweiligen Paket dokumentiert werden.
- Das **Manifest** beinhaltet Prüfsummen sämtlicher zu einem Paket gehörender Dateien sowie der Quell-Archive. Es wird zwischen den Formaten **Manifest** und **Manifest2** unterschieden, derzeit enthält der Portage Tree Prüfsummen in beiden Formaten – langfristig wird jedoch nur noch das **Manifest2** unterstützt werden.
- **metadata.xml** enthält analog zur **metadata.xml**-Datei der Kategorie eine kurze Beschreibung des Pakets. Darüber hinaus sind Informationen über die oder den jeweiligen Paketbetreuer enthalten.
- Das **files/**-Verzeichnis enthält kleinere Patches von maximal wenigen KB, die Fehler in den Quell-Archiven beheben. Zusätzlich sind Digest-Dateien enthalten, die Prüfsummen der Quell-Archive enthalten. Nach Abschluss der Umstellung auf das **Manifest2**-Format sind diese Digest-Dateien überflüssig.

Betrachten wir nun das Ebuild für die Version 0.1.25 der `portage-utils` – Sie sehen die Aufteilung in Metadaten einerseits und Anweisungen für die Kompilierung und Installation andererseits.

```
DESCRIPTION="small and fast portage helper tools written in C"
HOMEPAGE="http://www.gentoo.org/"
SRC_URI="mirror://gentoo/${P}.tar.bz2"

LICENSE="GPL-2"
SLOT="0"
KEYWORDS="~alpha ~amd64 ~arm ~hppa ~ia64 ~m68k ~mips ~ppc ~ppc64 ~s390 ~sh
~sparc ~sparc-fbsd ~x86 ~x86-fbsd"
IUSE=""
DEPEND=""
```

Listing 3.4: Metadaten-Bestandteil des `portage-utils-0.1.25.ebuild`

Neben für die Kompilierung und Installation des Pakets unwesentlichen Informationen wie der Homepage des Programms, Lizenz und Beschreibung sind essenzielle Informationen vorhanden:

- `SRC_URI`: Wo liegt das Quell-Archiv?
- `KEYWORDS`: Für welchen Zweig welcher Architekturen ist das Ebuild freigegeben?
- `IUSE`: Welche optionalen Komponenten unterstützt das Programm?
- `DEPEND`: Welche Abhängigkeiten benötigt das Programm?

In diesem sehr simplen Beispiel werden keine optionalen Komponenten unterstützt oder weitere Pakete als Abhängigkeit benötigt.

```
src_compile() {
    tc-export CC
    emake || die
}

src_install() {
    dobin q || die "do bin failed"
    doman man/*. [0-9]
    for applet in $(<applet-list) ; do
        dosym q /usr/bin/${applet}
    done
}
```

```
done  
}
```

Listing 3.5: portage-utils-0.1.25.ebuild

Die Shell-Funktionen `src_compile()` und `src_install()` übernehmen die bei der manuellen Kompilierung und Installation eines Programms notwendigen Schritte `./configure && make && make install`. Der `./configure && make`-Part wird über die Funktion `src_compile()` abgehandelt, `make install` über die Funktion `src_install()`.

3.4 Aktualisierung des Systems

Der wichtigste Modus – neben Installation und Deinstallation – von `emerge` ist die Aktualisierung eines Gentoo-Systems. Nachdem Sie die Installation mit einem Stage-Archiv begonnen haben, das aus einem Snapshot, einer Momentaufnahme, des Portage Tree heraus erstellt wurde, können zwischen Veröffentlichung des Stage-Archivs und Installation Ihres Gentoo-Systems mit diesem Stage-Archiv schnell einige Monate oder gar ein halbes Jahr vergehen. Innerhalb dieses Zeitraums sind zahlreiche Sicherheitsaktualisierungen, Korrekturen von Fehlern sowie zahlreiche neuere Versionen von Programmen in den Portage Tree eingepflegt worden. Um diese nutzen zu können, ist eine Aktualisierung des von Ihnen installierten Gentoo-Systems notwendig.

Die Aktualisierung gliedert sich in zwei Schritte:

1. Aktualisierung des Portage Tree
2. Installation aktualisierter Programme

3.4.1 Aktualisierung des Portage Tree

Erster Schritt bei der Aktualisierung eines Gentoo-Systems ist die Aktualisierung des Portage Tree – ohne neue Metadaten stehen keine neuen Pakete zur Verfügung.

Aktualisierung des Portage Tree

Die Aktualisierung des Portage Tree kann auf mehrere Arten erfolgen. Welche davon die für Sie passende ist, hängt von Ihrer Umgebung und insbesondere Einschränkungen durch etwaige Firewalls ab.

emerge --sync Die Synchronisierung des Portage Tree mit einem `rsync`-Mirror ist die vorgegebene Standardmethode. Diese Methode nutzt das `rsync`-Protokoll, bei dem die lokalen Datenbestände mit denen auf dem `rsync`-Mirror abgeglichen und Unterschiede übertragen werden. Durch die Nutzung eines inkrementellen Über-

tragungsverfahrens ist der Abgleich nicht nur schnell, sondern benötigt auch wenig Bandbreite.

Dieses Aktualisierungsverfahren ist jedoch nur anwendbar, wenn ausgehende TCP-Verbindungen zu Port 873 möglich sind, was in Unternehmensnetzwerken zumeist nicht der Fall sein wird.

Die Konfiguration des zu nutzenden `rsync`-Mirrors erfolgt über die Variable `SYNC` in der Datei `/etc/make.conf`.

```
# nano -w /etc/make.conf
SYNC="rsync://rsync.de.gentoo.org/gentoo-portage"
```

Die `rsync`-Mirror sind in verschiedenen DNS-Round-Robin-Rotationen organisiert. Die Hauptrotation ist via `rsync.gentoo.org` erreichbar, daneben gibt es `rsync.europe.gentoo.org` sowie länderspezifische DNS-Rotationen wie `rsync.de.gentoo.org` und `rsync.at.gentoo.org`.

```
# emerge --sync
```

emerge-websync Ist die Aktualisierung des Portage Tree über das `rsync`-Protokoll auf Grund restriktiver Firewall-Beschränkungen oder aus anderen Gründen nicht möglich, kann eine Aktualisierung über einen Snapshot des Portage Tree erfolgen.

Diese werden einmal täglich erzeugt und auf den verschiedenen Gentoo-Mirrorservern via HTTP oder FTP bereitgestellt. Hierbei fällt eine deutlich höhere Menge zu übertragender Daten an, da der gesamte Portage Tree, wenn auch in gepackter Form, übertragen werden muss. Die lokale Synchronisierung des Portage Tree findet ebenfalls via `rsync` statt, so dass zusätzlich temporär genügend freier Speicherplatz für zwei Kopien des Portage Tree vorhanden sein muss.

Die Konfiguration der zu nutzenden Gentoo-Mirrorserver erfolgt über die Variable `GENTOO_MIRRORS` in der `/etc/make.conf`.

```
# nano -w /etc/make.conf
GENTOO_MIRRORS="http://ftp.belnet.be/pub/mirrors/rsync.gentoo.org/gentoo"
```

In der Variablen können mehrere zu nutzende Gentoo-Mirrorserver deklariert werden, diese werden der Reihenfolge nach benutzt. Ist das gewünschte Archiv auf Mirrorserver A noch nicht verfügbar, wird versucht, das Archiv von Mirrorserver B herunterzuladen.

Hinweis

Eine Auflistung sämtlicher aktuell verfügbarer Gentoo-Mirrorserver finden Sie unter <http://www.gentoo.org/main/en/mirrors.xml>.

```
# emerge-webrsync
Fetching most recent snapshot
Attempting to fetch file dated: 20070320
portage-20070320.tar.bz2: OK
Syncing local tree...
scanning tarball...

[...]

*** Completed webrsync, please now perform a normal rsync if possible.
Update is current as of the of YYYYMMDD: 20070320
```

Listing 3.6: emerge-webrsync

emerge-delta-webrsync emerge-delta-webrsync funktioniert analog zu emerge-webrsync, jedoch mit dem Unterschied, dass nur die Differenz zwischen einem lokal vorhandenen Portage-Snapshot und dem aktuell auf einem Gentoo-Mirror verfügbaren Snapshot des Portage Tree übertragen werden muss.

Die Konfiguration der zu nutzenden Gentoo-Mirrorserver erfolgt über die Variable GENTOO_MIRRORS in der `/etc/make.conf`.

```
# nano -w /etc/make.conf
GENTOO_MIRRORS="http://ftp.belnet.be/pub/mirrors/rsync.gentoo.org/gentoo"
```

In der Variablen können mehrere zu nutzende Gentoo-Mirrorserver deklariert werden, diese werden der Reihenfolge nach benutzt. Ist das gewünschte Archiv auf Mirrorserver A noch nicht verfügbar, wird versucht, das Archiv von Mirrorserver B herunterzuladen.

Hinweis

Eine Auflistung sämtlicher aktuell verfügbarer Gentoo-Mirrorserver finden Sie unter <http://www.gentoo.org/main/en/mirrors.xml>.

emerge-delta-webrsync ist im Gegensatz zu **emerge-webrsync** nicht Bestandteil des Paketmanagementsystems, da es weitere Pakete (`diffball`, `tarsync`) als Abhängigkeiten benötigt. Es muss daher samt Abhängigkeiten zunächst installiert werden:

```
# emerge emerge-delta-webrsync
```

Nach der Installation steht das Skript zur Verfügung und kann ausgeführt werden:

```
homer ~ # emerge-del ta-webrsync
```

3.4.2 Installation aktualisierter Programme

Die Aktualisierung bereits installierter Programme und Installation von eventuell neu hinzugekommenen Abhängigkeiten geschieht über ein `world` genanntes virtuelles Metapaket. Die Aktualisierung wird über den `--update`-Modus von `emerge` durchgeführt. Als Argument muss das zu aktualisierende Paket benannt werden, bei einer Aktualisierung des gesamten Systems `world`, bei Aktualisierung nur von Paketen aus dem Basissystem `system`.

Zusätzlich kann die Aktualisierung in zwei Modi durchgeführt werden. Der eine Modus aktualisiert alle explizit installierten Pakete, der andere Modus aktualisiert alle installierten Pakete.

Aktualisierung explizit installierter Pakete

Die Aktualisierung explizit installierter Pakete umfasst sämtliche Pakete, die von Ihnen zur Installation angegeben wurden. Als Beispiel der GNOME-Desktop – Sie geben zur Installation das Paket `gnome` an, als Abhängigkeiten von diesem Metapaket werden alle Komponenten des GNOME-Desktops mit installiert. In die Datei installierter Pakete, die so genannte »World«-Datei, wird jedoch nur das Metapaket `gnome` eingetragen. Im einfachen Aktualisierungsmodus werden nun alle in der World-Datei vorhandenen Pakete sowie Pakete, die zum Basissystem gehören, auf vorhandene neuere Versionen überprüft.

Vor der Aktualisierung bietet `emerge` die Möglichkeit, ein »Was-wäre-wenn« anzuzeigen, eine Auflistung aller Programme und Bibliotheken, die als Abhängigkeit des zu installierenden Programms zwingend benötigt werden. Erreicht wird dies über die Option `--pretend`, frei übersetzt »so tun als ob«.

```
# emerge --update world --pretend
These are the packages that would be merged, in order:

Calculating world dependencies... done!
[ebuild U ] sys-apps/man-1.6d [1.6-r1]
[...]
[ebuild N ] x11-proto/xf86bigfontproto-1.1.2 USE="-debug"
[ebuild N ] x11-proto/inputproto-1.3.2 USE="-debug"
[ebuild N ] x11-proto/bigreqsproto-1.0.2 USE="-debug"
[ebuild N ] x11-proto/xcmiscproto-1.1.2 USE="-debug"
[ebuild N ] x11-libs/libXau-1.0.2 USE="-debug"
```

```
[ebuild N   ] x11-libs/libXdmp-1.0.1 USE="-debug"  
[...]  
[ebuild   U ] sys-libs/db-4.2.52_p4-r2 [4.2.52_p2-r1] USE="-tc1% -test%"  
[ebuild   U ] dev-lang/python-2.4.3-r4 [2.4.3-r1] USE="-tk%"  
[ebuild   U ] sys-apps/file-4.19 [4.17-r1]
```

Listing 3.7: Auflistung zu aktualisierender Programme

Neben zu aktualisierenden Programmen stehen in diesem Beispiel auch Anwendungen zur Installation an, da diese als Abhängigkeit neuerer Programmversionen benötigt werden. Nach Prüfung der anstehenden Aktualisierungen können Sie die Aktualisierung nun anstoßen:

```
# emerge --update world  
[...]  
--- !empty dir /usr  
--- !empty dir /sbin  
--- !empty dir /etc  
--- !empty dir /bin  
>>> Auto-cleaning packages...  
  
>>> No outdated packages were found on your system.  
* GNU info directory index is up-to-date.  
* IMPORTANT: 25 config files in '/etc' need updating.  
* Type emerge --help config to learn how to update config files.
```

Listing 3.8: Durchführung der System-Aktualisierung

Nach der Durchführung der Aktualisierung aller explizit installierten Pakete wird Ihnen in den meisten Fällen der Hinweis auf aktualisierte, aber noch zurückgehaltene Konfigurationsdateien angezeigt werden. Die Funktion dieses Schutzes von Konfigurationsdateien und der Umgang mit aktualisierten Konfigurationsdateien wird in Kapitel 6 beschrieben.

Wichtig

Bei Nutzung von `emerge --update world` zur Aktualisierung der installierten Pakete werden Pakete, bei denen sicherheitsrelevante Aktualisierungen notwendig sind, nicht zwangsläufig in die Liste zu aktualisierender Anwendungen mit aufgenommen. Sie sollten zusätzlich zwingend regelmäßig das in Kapitel 10 beschriebene Programm `glsa-check` zur Überprüfung auf vorliegende sicherheitsrelevante Aktualisierungen nutzen.

Aktualisierung aller installierten Pakete

Zur Aktualisierung sämtlicher installierter Pakete verfügt der Aktualisierungsmodus über den zusätzlichen Schalter `--deep`. Das Anhängen des `--deep`-Parameters führt dazu, dass das Paketmanagement bei Aktualisierungen sowohl Pakete des Basissystems als auch explizit installierte und in der World-Datei festgehaltene sowie zusätzlich deren Abhängigkeiten berücksichtigt.

```
# emerge --update --deep world --pretend

These are the packages that would be merged, in order:

Calculating world dependencies... done!
[ebuild U ] sys-devel/gnuconfig-20060702 [20060227]
[ebuild U ] sys-kernel/linux-headers-2.6.17-r2 [2.6.11-r2] USE="-gcc64%"
[ebuild U ] app-misc/pax-utils-0.1.15 [0.1.13]
[ebuild U ] dev-libs/gmp-4.2.1-r1 [4.2.1]
[ebuild U ] sys-libs/timezone-data-2007c [2006a] USE="nls%"
[ebuild U ] sys-apps/busybox-1.4.1-r2 [1.1.3]
[ebuild U ] app-arch/bzip2-1.0.4 [1.0.3-r6]
[...]
```

Listing 3.9: Aktualisierung aller installierten Pakete

Wird `emerge` dann noch in den `--verbose`-Modus versetzt, zeigt es nach Prüfung aller Aktualisierungen auch die Gesamtzahl zu aktualisierender und neu zu installierender Pakete sowie die Gesamtgröße der benötigten Quell-Archive an.

```
# emerge --update --deep world --pretend --verbose

These are the packages that would be merged, in order:

Calculating world dependencies... done!
[ebuild U ] sys-devel/gnuconfig-20060702 [20060227] 39 kB
[ebuild U ] sys-kernel/linux-headers-2.6.17-r2 [2.6.11-r2] USE="-gcc64%" 40,347 kB
[ ... ]
[ebuild U ] sys-apps/shadow-4.0.18.1 [4.0.15-r2] USE="cracklib% nls pam -nousuid (-selinux) -skey" 1,481 kB
```

```
[ebuild U ] net-misc/openssh-4.5_p1-r1 [4.3_p2-r1] USE="X%* pam tcpd -
X509 -chroot -hpn -kerberos -ldap -libedit (-selinux) -skey -smartcard -
static (-ipv6%*) (-sftplogging%)" 944 kB
```

```
Total: 74 packages (54 upgrades, 19 new, 1 in new slot), Size of down-
loads: 164,993 kB
```

Listing 3.10: Aktualisierung aller Pakete, --verbose-Modus

Nach Prüfung der zu aktualisierenden Pakete kann die Aktualisierung nun gestartet werden.

```
# emerge --update --deep world
[...]
--- !empty dir /usr
--- !empty dir /sbin
--- !empty dir /etc
--- !empty dir /bin
>>> Auto-cleaning packages...

>>> No outdated packages were found on your system.
* GNU info directory index is up-to-date.
* IMPORTANT: 25 config files in '/etc' need updating.
* Type emerge --help config to learn how to update config files.
```

Listing 3.11: Durchführung der System-Aktualisierung

Nach der Durchführung der Aktualisierung aller explizit installierten Pakete wird Ihnen in den meisten Fällen der Hinweis auf aktualisierte, aber noch zurückgehaltene Konfigurationsdateien angezeigt werden. Die Funktion dieses Schutzes von Konfigurationsdateien und der Umgang mit aktualisierten Konfigurationsdateien wird in Kapitel 6 beschrieben.

3.5 Installation von Programmen

Die Installation von Programmen mit Gentoo's Portage geht einfach und intuitiv vonstatten. Die Auflösung von Abhängigkeiten, das heißt Programmen und Bibliotheken, die für die Kompilierung oder Ausführung eines Programms erforderlich sind, geschieht automatisch, ohne dass ein Eingriff durch den Benutzer notwendig wird.

Tipp

Für den Aufruf nahezu aller Optionen und Modi des Paketmanagements gibt es neben der langen, »sprechenden« Bezeichnung auch einen kurzen Parameter, bei `--pretend` beispielsweise `-p`. Zur besseren Veranschaulichung des Einstiegs in das Paketmanagement werden folgend zunächst die sprechenden Bezeichnungen genutzt.

Die Veranschaulichung der Installations- und Deinstallationsmodi findet am Beispiel des Texteditors vim statt.

```
# emerge vim --pretend
These are the packages that would be merged, in order:

Calculating dependencies... done!
[ebuild N ] dev-util/ctags-5.5.4-r2
[ebuild N ] app-admin/eselect-1.0.7 USE="-bash-completion -doc"
[ebuild N ] app-admin/eselect-vi-1.1.4
[ebuild N ] app-editors/vim-core-7.0.174 USE="nls -acl -bash-comple-
tion -livecd"
[ebuild N ] app-editors/vim-7.0.174 USE="gpm nls perl python -acl -
bash-completion -cscope -minimal -ruby -vim-pager -vim-with-x"
[ebuild N ] app-vim/gentoo-syntax-20051221-r1 USE="-ignore-glep31"
```

Listing 3.12: Installation des Texteditors vim, `--pretend`-Modus

Aus dieser Ausgabe lassen sich folgende Informationen ablesen:

1. Alle Programme sind bisher nicht installiert und werden neu in das System installiert.
2. Neben vim werden ctags, eselect, eselect-vi, vim-core und gentoo-syntax benötigt.
3. vim hat als einziges Programm aktivierte USE-Flags (gpm, nls, perl und python).

Weitere Informationen über die Installation von Programmen lassen sich gewinnen, wenn emerge in einen geschwätzigeren Modus versetzt wird. Dies wird über die zusätzliche Option `--verbose` erreicht.

```
# emerge vim --pretend --verbose
These are the packages that would be merged, in order:
```

```
Calculating dependencies... done!  
[ebuild N   ] dev-util/ctags-5.5.4-r2 255 kB  
[ebuild N   ] app-admin/eselect-1.0.7 USE="-bash-completion -doc" 149 kB  
[ebuild N   ] app-admin/eselect-vi-1.1.4 2 kB  
[ebuild N   ] app-editors/vim-core-7.0.174 USE="nls -acl -bash-comple-  
tion  
livecd" 6,233 kB  
[ebuild N   ] app-editors/vim-7.0.174 USE="gpm nls perl python -acl -  
bash-completion -cscope -minimal -ruby -vim-pager -vim-with-x" 0 kB  
[ebuild N   ] app-vim/gentoo-syntax-20051221-r1 USE="-ignore-glep31" 19  
kB  
  
Total: 6 packages (6 new), Size of downloads: 6,656 kB
```

Listing 3.13: Installation des Texteditors vim, --pretend- und -verbose-Modi

Die zusätzliche Option `--verbose` führt im `--pretend`-Modus dazu, dass zusätzlich zur bisherigen Ausgabe auch die Größe der benötigten Quell-Archive angezeigt wird – sowohl für jedes einzelne als Abhängigkeit installierte Programm als auch in Summe für alle benötigten Quell-Archive.

Eine weitere Information, die aus beiden Ausgaben ersichtlich wird, sind die USE-Flags. Diese steuern die Installation optionaler Komponenten von Programmen und sind eines der Kern-Features des Paketmanagementsystems.

In diesem Beispiel würde vim als einziges zu installierendes Programm mit zusätzlichen Komponenten installiert – `gpm`, `nls`, `perl` und `python`. Die Idee hinter USE-Flags und Möglichkeiten zu deren Nutzung werden in Kapitel 5 detailliert vorgestellt.

Nachdem Sie nun Informationen zu den zusätzlich zum Texteditor vim benötigten Programmen eingesehen haben, können Sie das Programm installieren. Das Paketmanagementsystem übernimmt sämtliche erforderlichen Schritte – angefangen vom Herunterladen und Auspacken der Quell-Archive bis hin zu Kompilierung und Installation in das Gentoo-System.

```
# emerge vim
```

3.6 Deinstallation von Programmen

Genauso einfach wie die Installation von Programmen kann die Deinstallation von Programmen über das Paketmanagementsystem erfolgen. Mit einem Unterschied: Im Gegensatz zur Installation erfolgt bei der Deinstallation keine Auflösung von

Abhängigkeiten. Das heißt, nach der Deinstallation von Programmen sind als deren Abhängigkeit installierte Programme nach wie vor installiert und befinden sich möglicherweise in einem verwaisten Status, wenn kein weiteres Programm diese benötigt.

Die Deinstallation von Programmen wird über die Option `--unmerge` gesteuert. Auch bei dieser Funktion ist die `--pretend`-Option verfügbar.

```
# emerge vim --unmerge --pretend

>>> These are the packages that would be unmerged:

app-editors/vim
  selected: 7.0.174
  protected: none
  omitted: none

>>> 'Selected' packages are slated for removal.
>>> 'Protected' and 'omitted' packages will not be removed.
```

Listing 3.14: Deinstallation des Texteditors vim, `--unmerge`- und `--pretend`-Modi

In diesem Beispiel würde die Version 7.0.174 des Texteditors vim deinstalliert werden:

```
# emerge vim --unmerge

app-editors/vim
  selected: 7.0.174
  protected: none
  omitted: none

>>> 'Selected' packages are slated for removal.
>>> 'Protected' and 'omitted' packages will not be removed.

>>> Waiting 5 seconds before starting...
>>> (Control-C to abort)...
>>> Unmerging in: 5 4 3 2 1
>>> Unmerging app-editors/vim-7.0.174...
No package files given... Grabbing a set.
```

```
<<<    sym /usr/bin/vimdiff
<<<    obj /usr/bin/vim
<<<    sym /usr/bin/rvim
<<<    sym /usr/bin/rview
--- !empty dir /usr/bin
--- !empty dir /usr
* No suitable vim binary to rebuild documentation tags
* Calling eselect vi update...
* GNU info directory index is up-to-date.
```

Listing 3.15: Deinstallation des Texteditors vim

Bevor die tatsächliche Deinstallation erfolgt, sehen Sie noch einmal die Ausgabe des `--pretend`-Modus und haben fünf Sekunden Zeit, die Deinstallation abzubrechen, für den Fall, dass Sie sich verschrieben oder den Deinstallationsmodus aus Versehen gewählt haben.

Die weiteren zuvor als Abhängigkeit mitinstallierten Programme `ctags`, `eselect`, `eselect-vi`, `vim-core` und `gentoo-syntax` sind nun nach wie vor noch installiert, obwohl keine anderen Programme eine Abhängigkeit zu diesen haben. Um auch diese Programme zu deinstallieren, steht der `--depclean`- (kurz für »Dependency Clean«) Modus zur Verfügung, dessen Nutzung jedoch problembehaftet sein kann. Dieses Problem betrifft im Besonderen (für das Paketmanagement) verwaiste Bibliotheken.

Auch wenn kein Programm innerhalb der Datenbank des Paketmanagements von diesen Bibliotheken abhängt, können noch Programme dynamisch gegen diese gelinkt sein. Ergebnis der Deinstallation dieser Bibliotheken wäre, dass solche Programme nicht weiter ausführbar wären. Auslöser wäre ein Fehler im Paketmanagement, in dem Paket zu dem betreffenden Programm müsste die Abhängigkeit notiert sein – folglich würde die Bibliothek bei der Ausführung des `--depclean`-Modus übersprungen werden.

Bei der Installation des Texteditors vim sind jedoch keine Bibliotheken installiert worden, so dass der `--depclean`-Modus an dieser Stelle gefahrlos genutzt werden kann. Auch hier zeigt der `--pretend`-Modus analog zur Deinstallation einzelner Programme detailliert an, welche Versionen von Programmen zur Deinstallation vorgesehen sind. Hier, wie erwartet, die Programme `ctags`, `eselect`, `eselect-vi`, `vim-core` und `gentoo-syntax`:

```
# emerge --depclean --pretend
[...]
```

```
Calculating dependencies... done!

>>> These are the packages that would be unmerged:

dev-util/ctags
  selected: 5.5.4-r2
  protected: none
  omitted: none

app-admin/eselect
  selected: 1.0.7
  protected: none
  omitted: none

app-vim/gentoo-syntax
  selected: 20051221-r1
  protected: none
  omitted: none

app-editors/vim-core
  selected: 7.0.174
  protected: none
  omitted: none

app-admin/eselect-vi
  selected: 1.1.4
  protected: none
  omitted: none

>>> 'Selected' packages are slated for removal.
>>> 'Protected' and 'omitted' packages will not be removed.

Packages installed: 106
Packages in world: 1
Packages in system: 56
Unique package names: 106
Required packages: 100
Number to remove: 5
```

Listing 3.16: Deinstallation nicht weiter benötigter Abhängigkeiten, --pretend-Modus

Nach der tatsächlichen Deinstallation dieser Programme sind nun alle als Abhängigkeit von vim installierten Programme wieder vollständig aus dem System entfernt, wie Ihnen nach der Deinstallation noch einmal angezeigt wird. Aus »Number to remove« wurde »number removed« – die Anzahl der deinstallierten Programme.

```
# emerge --depclean
[...]
```

Packages installed:	106
Packages in world:	1
Packages in system:	56
Unique package names:	106
Required packages:	100
Number removed:	5

Listing 3.17: emerge --depclean, Anzahl deinstallierter Programme

3.7 Suchen von Paketen

Sei es eine Benennung, die von der Benennung eines Programms durch die Upstream-Autoren abweicht, die Verschiebung eines Programms in eine andere Kategorie innerhalb des Portage Tree oder aber das pure Interesse, wie viele Pakete in der Gentoo-Distribution auf einen Suchstring passen – Gentoo's Paketmanagement hilft Ihnen auch, ein Paket im Portage Tree zu finden. Neben den in das Paketmanagement integrierten Suchfunktionen gibt es weitere Programme, die – da auf einem Cache basierend – zum Teil schneller arbeiten.

3.7.1 Suche mit dem Paketmanagement

Das Paketmanagement Portage bietet zwei integrierte Modi zur Suche von Paketen innerhalb des Portage Tree.

- `emerge --search` erlaubt die Suche nach Paketnamen.
- `emerge --searchdesc` bezieht die Beschreibungen der jeweiligen Pakete in die Suche mit ein.

Als Beispiel für die Nutzung der Suchfunktion dient das Programm SpamAssassin. Von den Upstream-Autoren SpamAssassin benannt, ist das Paket in der Gentoo-Distribution in der Kategorie `mail-filter` als `spamassassin` verfügbar. Der Versuch, das Paket SpamAssassin zu installieren, schlägt daher fehl.

Wichtig ist an dieser Stelle zu wissen, dass Portage bei der Installation von Paketen zwischen Groß- und Kleinschreibung unterscheidet. In diesem Fall gibt es zwar ein Paket `spamassassin`, wie die folgende Suche zeigen wird, nicht jedoch ein Paket `SpamAssassin`.

```
# emerge SpamAssassin --pretend

These are the packages that would be merged, in order:

Calculating dependencies
emerge: there are no ebuilds to satisfy "SpamAssassin".
```

Die Suche nach `SpamAssassin` führt nun das Paket `spamassassin` und weitere auf den Suchstring passende Pakete auf.

```
# emerge --search SpamAssassin
Searching...
[ Results for search key : SpamAssassin ]
[ Applications found : 5 ]

* mail-filter/spamassassin
  Latest version available: 3.2.0-r1
  Latest version installed: 3.2.0-r1
  Size of files: 1,042 kB
  Homepage:      http://spamassassin.apache.org/
  Description:   SpamAssassin is an extensible email filter which is
  used to identify spam.
  License:       Apache-2.0

* mail-filter/spamassassin-fuzzyocr
  Latest version available: 3.5.1
  Latest version installed: [ Not Installed ]
  Size of files: 121 kB
  Homepage:      http://fuzzyocr.own-hero.net/
  Description:   SpamAssassin plugin for performing Optical Character
  Recognition (OCR) on attached images
  License:       Apache-2.0

[ ... ]
```

Listing 3.18: Suche nach dem Paket `SpamAssassin`

Neben dem Paket `spamassassin` hat die vorangegangene Suche auch das Plug-In `SpamAssassin-FuzzyOCR` hervorgebracht. Wollen Sie nun nach weiteren Plug-Ins für `SpamAssassin` oder mit `SpamAssassin` nutzbaren Paketen suchen, können Sie die Beschreibungen der einzelnen Pakete mit in die Suche einbeziehen.

```
# emerge --searchdesc SpamAssassin
Searching...
[ Results for search key : SpamAssassin ]
[ Applications found : 7 ]

* mail-filter/spamassassin
  Latest version available: 3.2.0-r1
  Latest version installed: 3.2.0-r1
  Size of files: 1,042 kB
  Homepage:      http://spamassassin.apache.org/
  Description:   SpamAssassin is an extensible email filter which is
used to identify spam.
  License:      Apache-2.0
[...]
```

```
* mail-filter/clamassassin
  Latest version available: 1.2.4
  Latest version installed: [ Not Installed ]
  Size of files: 34 kB
  Homepage:      http://jameslick.com/clamassassin/
  Description:   clamassassin is a simple script for virus scanning
(through clamav) an e-mail message as a filter (like spamassassin)
  License:      BSD
```

```
* mail-filter/spamass-milter
  Latest version available: 0.3.1-r1
  Latest version installed: [ Not Installed ]
  Size of files: 113 kB
  Homepage:      http://savannah.nongnu.org/projects/spamass-milt/
  Description:   A Sendmail milter for SpamAssassin
  License:      GPL-2
```

Listing 3.19: Suche nach `SpamAssassin` unter Einbeziehung der Paketbeschreibung

Neben dem Paket `spamassassin` und den anderen Ergebnissen aus der vorherigen Suche bringt die Einbeziehung der Paketbeschreibungen zwei weitere Ergebnisse – die Pakete `clamassassin` und `spamass-milter`.

3.7.2 Weitere Suchwerkzeuge

Wie Sie festgestellt haben, führt die in das Paketmanagement integrierte Suche zwar zum Ziel, ist mitunter jedoch recht träge und zeitintensiv. Die integrierten Suchmodi greifen direkt auf den Portage Tree zu, das heißt, jedes Ebuild muss zum Lesen geöffnet werden – bei der Masse der vorhandenen Ebuilds erklärt sich der benötigte Zeitaufwand.

Mit `esearch` und `eix` stehen zwei zusätzliche Programme bereit, die dieses Manko umgehen, indem sie einen Cache benutzen. Dieser muss jedoch nach jeder Aktualisierung des Portage Tree aktualisiert werden, damit die Suche auf aktuell vorhandenen Paketen aufbauen kann.

`esearch`

`esearch` ist als separates Paket verfügbar und muss daher zunächst installiert werden.

```
# emerge esearch
```

Da `esearch` auf einem eigenen Cache aufbaut, muss dieser über das Hilfsprogramm `eupdatedb` zunächst erstellt werden. Beachten Sie, dass der Cache nach jeder Aktualisierung des Portage Tree erneut erstellt werden muss, um die Aktualität der Datenbasis zu gewährleisten.

```
# esearch spamassassin
* Error: Could not find esearch-index. Please run eupdatedb as root first
# eupdatedb
* indexing: 0 ebuilds to go
* esearch-index generated in 30 second(s)
* indexed 11695 ebuilds
* size of esearch-index: 1808 kB
```

Nach dem Erstellen des Cache ist `esearch` nun bereit, genutzt zu werden. `esearch` unterstützt mehrere Modi, darunter einen Standmodus, der mit `emerge --search` identisch ist, und einen mit `emerge --searchdesc` identischen Modus `--searchdesc`.

```
# esearch SpamAssassin
[ Results for search key : SpamAssassin ]
[ Applications found : 5 ]

* mail-filter/spamassassin
  Latest version available: 3.2.0-r1
  Latest version installed: 3.2.0-r1
  Size of downloaded files: [no/bad digest]
  Homepage:   http://spamassassin.apache.org/
  Description: SpamAssassin is an extensible email filter which is used
to identify spam.
  License:    Apache-2.0
[...]
```

Listing 3.20: Suche nach SpamAssassin mit esearch

Analog zu `emerge --searchdesc SpamAssassin` liefert `esearch --searchdesc SpamAssassin` ebenfalls zwei weitere Ergebnisse. Zur genaueren Eingrenzung der Suchergebnisse kann der Suchstring auch als regulärer Ausdruck angegeben werden.

Ein weiterer interessanter Modus von `esearch` ermöglicht die Eingrenzung der Suche auf installierte Pakete, hier am Beispiel von KDE verdeutlicht.

```
# esearch --instonly kde
[ Results for search key : kde ]
[ Applications found : 3 ]

* kde-base/kdelibs
  Latest version available: 3.5.6-r8
  Latest version installed: 3.5.6-r8
  Size of downloaded files: [no/bad digest]
  Homepage:   http://www.kde.org/
  Description: KDE libraries needed by all KDE programs.
  License:    GPL-2 LGPL-2

* kde-base/kdesu
  Latest version available: 3.5.6-r1
  Latest version installed: 3.5.6-r1
  Size of downloaded files: [no/bad digest]
```

```
Homepage:  http://www.kde.org/  
Description: KDE: gui for su(1)  
License:   GPL-2  
  
* kde-base/libkdegames  
  Latest version available: 3.5.6  
  Latest version installed: 3.5.6  
  Size of downloaded files: [no/bad digest]  
  Homepage:  http://www.kde.org/  
  Description: Base library common to many KDE games.  
  License:   GPL-2
```

Listing 3.21: Beschränkung der Suche mit `esearch` auf installierte Pakete

eix

Genau wie `esearch` ist auch `eix` ein zusätzliches Paket, das vor der Nutzung zunächst installiert werden muss. Ebenfalls ist es erforderlich, einen Cache anzulegen.

```
# emerge eix  
# update-eix  
Reading Portage settings ..  
Building database (/var/cache/eix) ..  
[0] /usr/portage/ (cache: metadata)  
  Reading 100%  
Applying masks ..  
Database contains 11666 packages in 149 categories.
```

Listing 3.22: Installation von `eix` und Anlegen des Cache

Bei der Suche nach Paketen bietet `eix` einen unschlagbaren Vorteil, der Suchstring kann reguläre Ausdrücke beinhalten. Ganz rudimentär lässt sich die Suche so beispielsweise auf Pakete begrenzen, auf die der exakte Suchstring zutrifft.

```
# eix ^spamassassin$  
[I] mail-filter/spamassassin  
  Available versions: 3.1.8 3.1.8-r1 (~)3.2.0 (~)3.2.0-r1 {berkdb doc  
  ipv6 ldap mysql postgres qmail sqlite ssl tools}  
  Installed versions: 3.2.0-r1(15:58:03 08.05.2007) (berkdb -doc -ipv6 -  
  ldap -mysql -postgres -qmail -sqlite ssl -tools)
```

```
Homepage:      http://spamassassin.apache.org/
Description:   SpamAssassin is an extensible email filter which
is used to identify spam.
```

Listing 3.23: Nutzung der Suchfunktion von `eix`

Ein weiterer Unterschied ist die Anzeige der Suchergebnisse. Während die in das Paketmanagement integrierten Suchfunktionen und `esearch` eine nahezu identische Ausgabe produzieren, setzt sich `eix` hier ab und gibt mehr Informationen aus. So werden für jede vorhandene Version die Zugehörigkeit zu einem Zweig der Distribution (stabil, testing, maskiert) und die unterstützten USE-Flags angezeigt. Zu der installierten Version werden neben dem Installationsdatum auch die bei der Installation genutzten USE-Flags dargestellt.

Ein Alleinstellungsmerkmal von `eix` ist die Anzeige nach Aktualisierung des programminternen Cache – hier werden neu in die Distribution aufgenommene oder entfernte Programme oder Programme, die in den stabilen Zweig verschoben wurden, gesondert angezeigt.

```
# update-eix
* Running update-eix
Reading Portage settings ..
Building database (/var/cache/eix) ..
[0] /usr/portage/ (cache: metadata)
    Reading 100%
Applying masks ..
Database contains 11664 packages in 149 categories.
Diffing databases (11666 - 11664 packages)
[>] == app-crypt/gnupg (1.4.6 1.9.21(1.9) -> 1.4.7-r1 1.9.21(1.9)): The GNU Pri-
vacy Guard, a GPL pgp replacement
[>] == app-crypt/gpgme (0.3.14-r1(0.3) 1.1.2-r1(1) -> 0.3.14-r1(0.3) 1.1.4(1)):
GnuPG Made Easy is a library for making GnuPG easier to use
[U] == dev-lang/php (5.2.1-r3(5)@04/09/07; 4.4.6(4) 5.2.1-r3(5) -> 4.4.7(4)
5.2.2-r1(5)): The PHP language runtime engine: CLI, CGI and Apache2 SAPIs.
    << mail-client/mahogany ([M]!0.65): Highly customizable powerful mail client
[U] == sys-libs/timezone-data (2007c@04/09/07; 2007c -> 2007e): Timezone data
(/usr/share/zoneinfo) and utilities (tzselect/zic/zdump)
[...]
```

Listing 3.24: Anzeige der Änderungen nach Aktualisierung des Portage Tree

In diesem Beispiel liegen die nicht installierten Anwendungen `gnupg` und `gpgme` sowie die installierten Pakete `php` und `timezone-data` in neuen Versionen vor. Das Paket `mahogany` hingegen wurde aus dem Portage Tree entfernt.

`eix` beinhaltet einen Wrapper, über den sowohl die Aktualisierung des Portage Tree als auch die Aktualisierung des `eix`-Cache in einem vorgegebenen Zeitraum vorgenommen werden kann.

```
# eix-sync
```

3.8 Grafische Frontends

Neben dem Kommandozeilenclient `emerge` gibt es einige weitere grafische Frontends zum Paketmanagementsystem Portage. `Porthole`, `Kuroo` und `portageMaster` bedienen jeweils ihre eigene Klientel.

`portageMaster` eignet sich zum schnellen Anzeigen von Informationen, die beiden »mächtigeren« Anwendungen `Kuroo` und `Porthole` sind auf die verbreiteten Desktopsysteme KDE und GNOME zugeschnitten.

Zu beachten ist, dass die grafischen Frontends durch eigene Projekte außerhalb von Gentoo entwickelt werden. Sie werden im Gegensatz zu `emerge` nicht als Teil des Paketmanagementsystems entwickelt und gepflegt. Daher ist eine vollständige Funktion dieser Programme nicht immer garantiert. Dies wird sich jedoch voraussichtlich mit Festlegung der EAPI ändern, in der Funktionalität und Verhalten des Paketmanagementsystems festgeschrieben werden. Sollten Sie auf spezielle Probleme mit diesen Frontends treffen, so ist dies mit großer Wahrscheinlichkeit ein Fehler in der Anwendung – Sie sollten den Programmierern den Fehler möglichst detailliert mitteilen.

3.8.1 Porthole

Die Frontends `Porthole` und `Kuroo` verfügen in etwa über den gleichen Umfang, sie ermöglichen über die Funktionalitäten von `portageMaster` hinaus die selektive Installation einzelner Pakete, die Suche innerhalb des Portage Tree und verfügen nicht zuletzt über eine ansprechendere Benutzeroberfläche.

`Porthole` bietet zusätzlich die Möglichkeit, Befehle vorzudefinieren und zur Installation von Paketen zu nutzen. Die Entwicklung der Anwendung scheint Ende 2005 mit der aktuellen Version 0.5.0 stehen geblieben zu sein – in den Vorgaben zur Definition individueller Kommandos werden noch längst als veraltet gekennzeichnete Funktionen, wie etwa `ACCEPT_KEYWORDS=""~x86"` `emerge pakename`, vorgegeben.

Nachdem beim Start der Anwendung der Portage Tree initialisiert wurde, ermöglicht die Oberfläche direkt nach dem Start ein bequemes Browsen innerhalb des Portage Tree. Bereits installierte Anwendungen werden gesondert hervorgehoben.

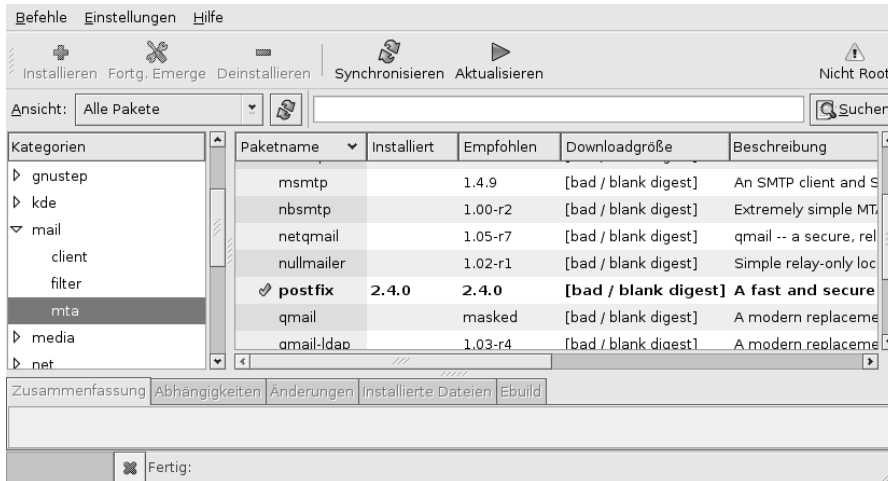


Abb. 3.5: Browsen durch den Portage Tree mit Porthole

Neben Funktionen zur Synchronisierung des Portage Tree und Aktualisierung des gesamten Systems können einzelne Anwendungen zur Installation ausgewählt werden – nachdem diese entweder durch Browsen durch den Portage Tree oder Suche nach dem Paketnamen ausgewählt wurden.

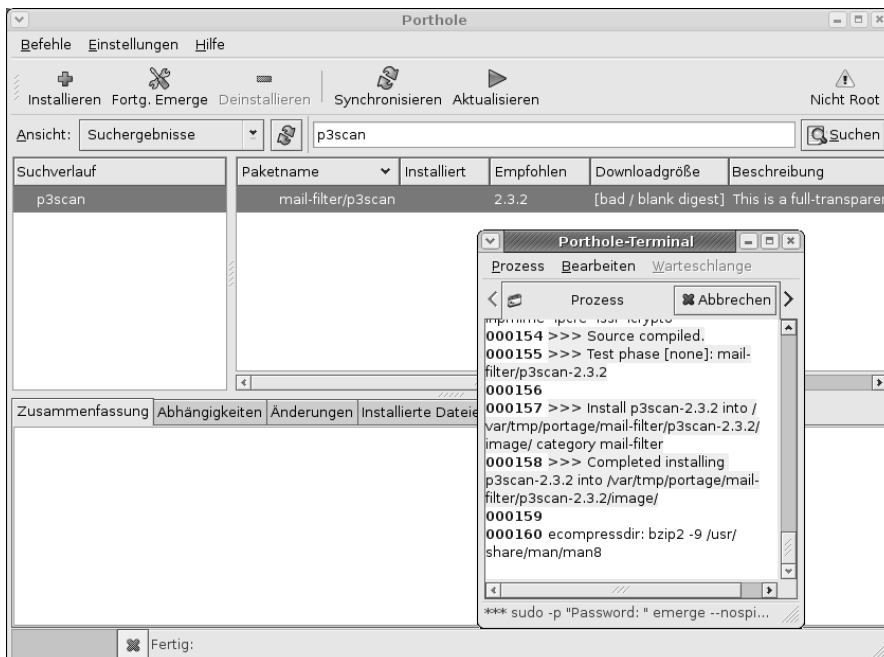


Abb. 3.6: Installation der Anwendung p3scan mit Porthole

3.8.2 Kuroo

Das Frontend Kuroo ist der persönliche Geheimtipp des Autors, die Anwendung verfügt über die meisten Funktionen aller verfügbaren Portage Frontends, hat eine sehr aufgeräumte und intuitive Benutzeroberfläche und wird aktiv weiterentwickelt.

Kernziele der Kuroo-Entwicklung sind die einfache Benutzbarkeit und das Funktionieren ohne vorheriges Eingreifen und Konfigurieren durch den Anwender. Ziel ist die Entwicklung einer stabilen Anwendung, die sich zur Erfüllung der grundlegenden administrativen Funktionen rund um Gentoo's Paketmanagement eignet.

Die Anzeige des Portage Tree ist auf zwei Spalten verteilt, um Kategorien wie `app`, `net` oder `media` und Subkategorien wie `sound`, `tv`, `radio` und `video` zeigen zu können. Die einzelnen Pakete werden mit Beschreibung in einer Tabelle angezeigt, für das in der Tabelle jeweils ausgewählte Paket werden zusätzlich weitere Informationen mit angezeigt, wie Abbildung 3.7 für das `apache`-Paket.

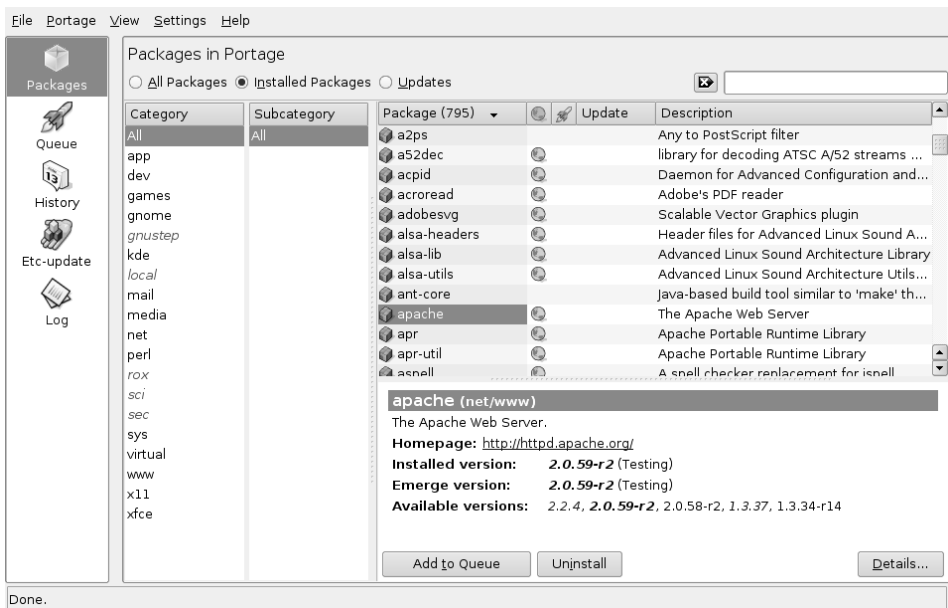


Abb. 3.7: Browsen des Portage Tree, Anzeige der Details zum Apache-Paket

Über den Button DETAILS lässt sich ein weiteres Fenster mit zahlreichen Informationen zu einem Paket öffnen. Dort lassen sich verfügbare Versionen, USE-Flags, das Change-Log, das Ebuild, Abhängigkeiten und – bei bereits installierten Paketen – auch die Dateien eines Pakets anzeigen.

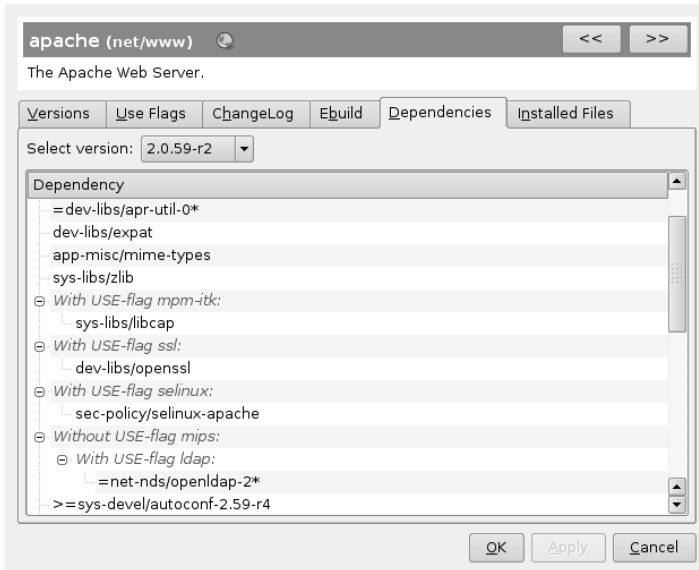


Abb. 3.8: Detailansicht zum Apache-Paket

Installation von Anwendungen mit Kuroo

Die Installation von Paketen mit Kuroo geschieht in zwei, genau genommen in drei Schritten. Zunächst können aus der Übersicht einzelne Pakete in die Installationswarteschlange eingereiht werden.

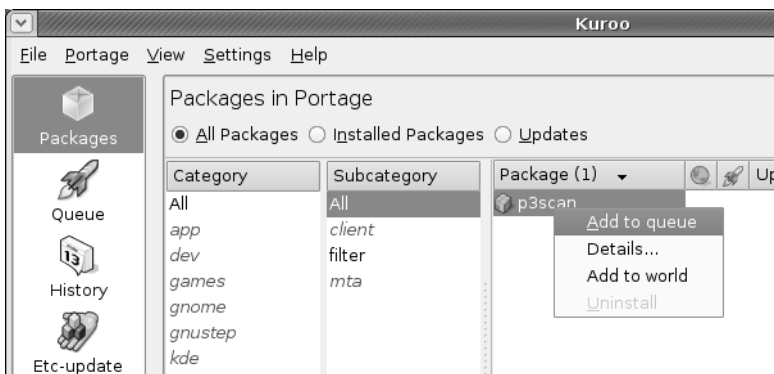


Abb. 3.9: Einreihen von Anwendungen in die Installationswarteschlange

Die eigentliche Installation der Anwendung erfolgt in zwei weiteren Schritten, nachdem Sie in der Funktionsliste auf die Installationswarteschlange (Queue) zugegriffen haben.

- In Schritt 1 überprüfen Sie die zur Installation vorgemerkten Pakete. Hierbei wird `emerge` im `--pretend`-Modus aufgerufen, um evtl. nicht korrekte Abhängigkeiten auflösen zu können.
- In Schritt 2 wird die eigentliche Installation durchgeführt. Nach der Installation werden die von Portage ausgegebenen Meldungen angezeigt.

Eine Besonderheit bei Kuroo ist die Anzeige von Fortschrittsbalken bei Installationen. Hier wird auf zurückliegende Installationen eines Pakets zugegriffen und aus der Dauer der Installation dieses Pakets ein Mittelwert gebildet, der Grundlage für die Berechnung des Fortschrittsbalkens ist.

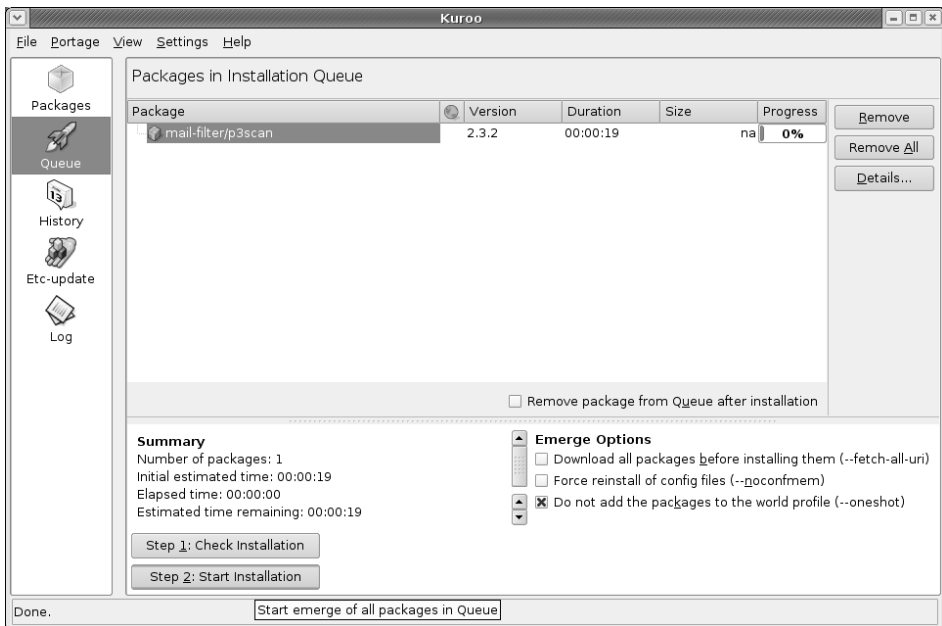


Abb. 3.10: Installation von Paketen mit Kuroo

3.9 Fazit

In diesem Kapitel haben Sie die grundlegenden Funktionen von `emerge` kennen gelernt.

Sie können mit dem Paketmanagementsystem nun Programme installieren und deinstallieren sowie eine vollständige Aktualisierung des Gentoo-Systems vornehmen. Die Einführung in den Portage Tree hat Ihnen gezeigt, wie Pakete innerhalb der Gentoo-Distribution organisiert sind.