# >> Gentoo Linux Init System

[Bitte Kapitel auswählen]

## 1. Einleitung

Gentoo Linux benutzt ein Init-System das größtenteils über Abhängigkeiten kontrolliert wird. Es sollte einfach zu warten und doch stark und flexibel genug sein für jede Art von Konfiguration. Diesen Text sollte man nicht als eine Einleitung in die inneren Mechanismen verstehen, sondern vielmehr als eine einfach Anleitung um mit Gentoo's Init-System arbeiten zu können. Leute die ernsthaft daran interessiert sind, die inneren Mechanismen zu verstehen .... lesen Sie den Quelltext ;-)

## 2. Runlevel

Im Gegensatz zu anderen Init-Systemen, bestehen Gentoos Runlevel nicht aus festen Namen oder Nummern, sondern vielmehr aus eigenen Namen, die auf die standard Runlevel von init abgebildet werden.
**Notiz**
Standardmäßig gibt es drei Runlevel, namentlich: **"boot"**, **"default"** und **"nonetwork"**.

Das "boot" Runlevel sollte der standard Typ für die meisten Setups sein, und ist, wie der Name sagt, das erste Runlevel das nach dem booten ausgeführt wird. Als nächstes kommt **"default"**, welches, wie der Name schon andeutet, das standard Runlevel ist und nach dem booten gestartet wird. Zuletzt folgt **"nonetwork"**, welches ausschließlich als Beispiel dient.

Die Runlevels liegen in /etc/runlevels, in einem Unterverzeichnis das nach dem Namen des Runlevels benannt wurde; dieses Unterverzeichnis enthält symbolische Links zu den Diensten, die in diesem Runlevel geladen werden sollen.
**Notiz**
Der bevorzugte Weg, um einen Service hinzuzufügen oder zu löschen wird später in dem Abschnitt "Über rc-update" behandelt.

Wie bereits erwähnt, kann der Name frei gewählt werden, solange die Datei /etc/inittab entsprechend dem neuen Namen des Runlevels angepasst wird.
**Wichtig**
Eine Ausnahme dieser Regel, die jedoch erwähnt werden sollte, stellt das Runlevel **"boot"** dar.
**Warnung**
Bitte ändern Sie den Namen des **"boot"** Runlevels NIEMALS, da es einige Dinge kaputt machen würde.

Die ganze Arbeit wird vom /sbin/rc Script erledigt, mit diesem können Sie auch im laufenden Betrieb zwischen den virtuellen Runleveln wechseln.

### 2.2 Virtuelle Runlevel

Da Runlevel nicht statisch auf die von Init gebunden sind, kann man wesentlich mehr haben als init unterstützt. Das ermöglicht es dem Benutzer nach seinen Bedürfnissen Profile und Virtuelle Runlevel zu erstellen

Zum Beispiel könnte ein Laptop Benutzer zwei standard Runlevel haben, "online" und "offline". Das würde erlauben ein aktives Runlevel zu benutzen, wenn die PCMCIA Netzwerkkarte eingesteckt ist, und ein weiteres Runlevel, wenn sie es nicht ist. Die PCMCIA Scripts könnten dann so konfiguriert werden, dass sie *"/sbin/rc online"* oder *"/sbin/rc offline"* aufrufen, um die richtigen Dienste zu starten oder zu stoppen, jeweils abhängig vom Status der PCMCIA Netzwerkkarte.

### 2.3 Runlevel und XFree86

Nach Gentoos Weg, Dinge zu erledigen haben wir kein Runlevel für X, sondern stattdessen ein startup-Script. Es hat den Namen "xdm" und kann zu jedem Runlevel hinzugefügt werden, wenn der Benutzer dieses wünscht.
**Notiz**
Dies sollte das hauptsächlich genutzte Runlevel des Nutzers sein.
**Warnung**
Es zum Boot Runlevel zu ergänzen kann in unerwünschten Nebeneffekten enden.

Wenn xdm, gdm oder kdm vor Ihren Gettys gestartet wird, startet X standardmäßig auf der nächsten freien Konsole. Auf langsameren Rechner ist das kein Problem, solange der Desktop Manager am Ende

des Runlevel Init Vorganges startet. Die Gettys werden vor X starten, welches dann auf Konsole 7 startet. Auf schnelleren Rechnern ist dies nicht der Fall. X wird hier vorher auf Konsole zwei starten. Wenn nun die Gettys starten, übernehmen diese die Kontrolle über die Tastatur, dadurch verliert der Desktop Manager die Kontrolle der Tastatur.

Dies wird dadurch verhindert, dass das Desktop Manager Startscript über ein "Extra-Runlevel", hier das Runlevel "a", geführt wird. Da das Runlevel "a" kein richtiges Runlevel ist, ruft unser "xdm" Script einfach *"telinit"* auf. Dadurch werden alle Dienste des Runlevels "a" erst nach dem aktuellen Runlevel gestartet, also nachdem die Gettys gestartet wurden.
**Notiz**
Sie können mehr Informationen über Runlevel "a" erhalten, indem Sie die man pages von init lesen.

# 3. RC-Scripts

Rc-Scripts sind Scripts, die die grundsätzlichen Funktionen, sowie die Abhängigkeiten von Diensten beim Start definieren. Sie liegen in /etc/init.d/.

## 3.2 Grundlayout eines RC-Skriptes

**Befehlsauflistung 1:** rc-script Layout

```
#!/sbin/runscript

depend() {
    need bar
}

start() {
    ebegin "Starting foo"
    /sbin/foo
    eend $? "Failed to start foo"
}

stop() {
    ebegin "Stopping foo"
    kill $(cat /var/run/foo.pid)
    eend $? "Failed to stop foo"
}
```

**Notiz**
Der Interpreter ist "/sbin/runscript".
**Notiz**
Die "depend" Funktion ist optional.
**Notiz**
Jedes rc-Script braucht mindestens die "start" Funktion.

## 3.3 Kontrollieren des Start-up

Die generelle Startreihenfolge der Dienste innerhalb eines Runlevels ist alphabetisch. Dies liegt an der Sortierung der Ausgabe von /bin/ls.

Die erste Möglichkeit von der Startreihenfolge abzuweichen sind Abhängigkeiten. Alternativ können auch die sogenannten "Order Types" genutzt werden.

# 4. Abhängigkeitstypen

Die meisten Dienste sind mit anderen Diensten verbunden und hängen sogar von ihnen ab.

Zum Beispiel benötigt Postfix ein laufendes Netzwerk, sowie einen Systemlogger.

Samba benötigt ebenfalls ein laufendes Netzwerk. Wenn CUPS zum Drucken benutzt werden soll, muss cupsd auf jeden Fall vor Samba gestartet werden. Beachten Sie, das CUPS nicht unbedingt zum Starten von Samba nötig ist.

Wir haben zwei verschiedene Möglichkeiten, um Abhängigkeiten zwischen verschiedenen Diensten zu definieren. Diese Abhängigkeiten sind immer gültig, egal ob ein Runlevel als ganzes gewechselt wird oder ein Service einfach nur manuell nach dem Booten gestartet wird.

## 4.2 Der NEED Abhängigkeitstyp

Dieser Typ wird benutzt, wenn ein Dienst für das Starten des aktuellen Dienstes dringend nötig ist.

**Befehlsauflistung 2:** Logger und net werden als NEED Abhängigkeit definiert

```
depend() {
    need net logger
}
```

**Notiz**
Die Dienste, welche nach **NEED** genannt werden, sind dringend erforderlich, damit der aktuelle Dienst starten kann. Der aktuelle Dienst wird also nicht starten, wenn eine der Abhängigkeiten nicht starten sollte.
**Wichtig**
Jeder Dienst, der in der **NEED** Zeile steht, wird gestartet, auch wenn dieser NICHT zum aktuellen oder **"boot"** Runlevel ergänzt wurde.
**NEED** ist deshalb eine "starke" Abhängigkeit.

## 4.3 Der USE Abhängigkeitstyp

Dieser Dienst ist nicht unbedingt zum Starten des aktuellen Dienstes nötig, sollte jedoch vor dem aktuellen gestartet werden.

**Befehlsauflistung 3:** Portmap wird als USE Abhängigkeit zu netmount ergänzt

```
depend() {
    use portmap
}
```

Netmount kann standardmäßig mit NFS Mounts umgehen, aber wird nur von Portmap abhängen, wenn dieses zum aktuellen oder zum Boot Runlevel ergänzt wurde. Jeder Benutzer, der NFS Mounts nutzt, sollte portmap zum Default Runlevel hinzufügen. Dadurch wird Portmap als USE Abhängigkeit gesehen und vor netmount gestartet werden.

**Wichtig**
Jeder Dienst in der **USE** Zeile ***muss*** im aktuellen oder im Boot Runlevel vorhanden sein, damit er als gültige **USE** Abhängigkeit anerkannt werden kann.
**USE** is dadurch eine "schwache" Abhängigkeit.

**Notiz**
Sollte ein Dienst in der **USE** Zeile nicht starten, so wird der aktuelle Dienst trotzdem gestartet werden, da der Dienst der **USE** Zeile nicht zwingend für den Start des aktuellen Dienstes nötig ist.

# 5. Kontrollieren der Reihenfolge ohne Abhängigkeiten

Sollte keine abhängige Verbindung zwischen zwei Diensten bestehen, es aber trotzdem nötig oder gewünscht sein, einen Dienst nach einem bestimmten anderen zu starten, können die **AFTER** und **BEFORE** Optionen genutzt werden.
**Notiz**
Diese beiden Typen sind nur während dem Wechseln eines Runlevels gültig.

Optional können diese beiden Typen ein "*" Wildcard für das integrieren aller anderen Dienste enthalten:

**Befehlsauflistung 4:** Ein Beispiel für AFTER

```
depend() {
    after *
}
```

Dies wird den Dienst ***nach*** allen anderen Diensten starten.

## 5.2 Die BEFORE Option

der aktuelle Dienst wird ***vor*** den in der **BEFORE** Zeile aufgelisteten Diensten gestartet.

**Befehlsauflistung 5:** Lässt foo vor dem Dienst bar starten (Auszug von foo)

```
depend() {
   before bar
}
```

### 5.3 Die AFTER Option

Der aktuelle Dienst wird ***nach*** den in der **AFTER** Zeile gelisteten Diensten gestartet.

**Befehlsauflistung 6:** Lässt bar nach foo starten (Auszug von bar)

```
depend() {
    after foo
}
```

## 6. Virtuelle Dienste

Dienste, wie die meisten Dinge in der heutigen Unix Welt, kommen in verschiedenen Farben und Geschmäckern daher. Normalerweise hat der Nutzer / Administrator die Wahl zu bestimmen welche genutzt werden.

Ein Beispiel dafür sind Systemlogger. Zum Zeitpunkt des Schreibens dieses Dokumentes, hat der Nutzer von Gentoo Linux die Auswahl aus vier verschiedenen. Alle Dienste, die einen Logger benötigen, können nicht alle vier mittels **NEED** Option anfordern. Die **USE** option ist zu schwach.

Dies ist der Punkt, an dem Virtuelle Dienste und die **PROVIDE** Option ins Spiel kommen.

### 6.2 Die PROVIDE Option

Die **PROVIDE** Option definiert einen virtuellen Dienst, welche andere Dienste mittels **NEED** und **USE** aufrufen können.

**Befehlsauflistung 7:** Sysklogd liefert logger

```
depend() {
    provide logger
}
```

### 6.3 Der Virtuelle Dienst LOGGER

**LOGGER** ist ein vordefinierter virtueller Dienst, welcher von allen Systemloggern geliefert wird. Er kann entweder mit **NEED** oder **USE** genutzt werden.

### 6.4 Der virtuelle Dienst NET

Der NET Dienst ist ein anderer virtueller Dienst, jedoch im Gegensatz zu **LOGGER** liefert er nicht einen eindeutigen Dienst.

**Wichtig**
Um den virtuellen Dienst **NET** zu unterstützen, muss ein Dienst:

- zum aktuellen oder Boot Runlevel angehören.
- ein "net" vorangestellt haben.
- der Teil nach "net" muss den Namen des eigentlichen Netzwerk Interfaces tragen (z.B. net.eth0 oder net.ppp1).

Für jeden gültigen net.* Dienst, wird $IFACE auf den Namen des Netzwerk Interfaces gesetzt(z.B. "eth0" für net.eth0).

## 7. Standard Kommandozeilen Optionen

Jeder Dienst kann mit einer der standard Optionen aufgerufen werden. All diese genannten sind bereits definiert, mit Ausnahme von **START** und **STOP**, welche dem Benutzer als Funktionen in seinem Rc-Script definieren sollte.
**Wichtig**
Die **start()** Funktion **muss** definiert werden.
**Notiz**
Die **stop()** Funktion ist nicht ganz so wichtig und kann weggelassen werden.

**Notiz**
In der Regel wird der Benutzer nur **start()**, **stop()** und **restart()** definieren. Der Rest wird intern ablaufen und sollte in Ruhe gelassen werden.

**Befehlsauflistung 8:** Start des HTTPD Dienstes

```
# /etc/init.d/httpd start
```

**Notiz**
Kommandozeilen Optionen können gestapelt, bzw. aufgereiht werden.

**Befehlsauflistung 9:** Pausieren / Starten von net.eth0

```
# /etc/init.d/net.eth0 pause start
```

## 7.2 Die START/STOP Option

**START**, startet den Dienst inklusive aller seiner Abhängigkeiten.

**STOP**, stoppt den Dienst inklusive aller Dienste, welche von ihm abhängig sind.

## 7.3 Die RESTART Option

Damit **RESTART** funktioniert, muss der Dienst vorher gestartet sein. Dadurch wird der Dienst und alle Dienste, die von ihm abhängen neu gestartet.
**Wichtig**
Sollte eine eigene **restart()** Funktion definiert sein, sollte der Nutzer zum Starten und Stoppen **"svc_start()"** und **"svc_stop()"** verwenden.
**Notiz**
Dies ist nötig, damit alle abhängigen Dienste richtig gehandhabt werden.

## 7.4 Die PAUSE Option

Dies wird den Dienst stoppen, nur im Gegensatz zu **STOP** wird kein abhängender Dienst angehalten.

## 7.5 Die ZAP Option

Setzt den Status eines Dienstes auf gestoppt.
**Notiz**
Beachten Sie, dass hier kein Kommando aus der **stop()** Funktion ausgeführt wird. Deshalb sollte der Benutzer alle nötigen "Aufräumarbeiten" manuell durchführen.

## 7.6 Die INEED/NEEDSME Optionen

**INEED** listet alle Dienste auf, die vom aktuellen Dienst gebraucht werden.

**NEEDSME** listet alle Dienste auf, die den aktuellen Dienst brauchen.

## 7.7 Die IUSE/USESME Optionen

**IUSE** listet alle Dienste auf, die der aktuelle Dienst benutzt (siehe USE Option / Typ weiter oben).

**USESME** listet alle Dienste auf, die den aktuellen Dienst nutzen (siehe USE Option / Typ weiter oben).

## 7.8 Die BROKEN Option

Diese listet alle fehlenden Dienste (falls es welche gibt) auf, die der aktuelle Dienst braucht.

# 8. Eigene Kommandozeilen Optionen ergänzen

Eigene Kommandozeilen Optionen zu ergänzen ist relativ einfach. Eine Funktion mit dem Namen der Option muss im Rc-Script definiert werden und zur **$opts** Variable ergänzt werden, siehe unten:

**Befehlsauflistung 10:** foo als eigene Option

```
opts="${opts} foo"

foo() {
    ............
}
```

# 9. Konfiguration

Konfiguration sollte generell durch Umgebungsvariablen erfolgen. Diese sollten *nicht* im Rc-Script definiert werden, sondern in einer der drei möglichen Konfigurations Dateien.

Eine, die speziell für das Rc-Script da ist und zwei globale Konfigurations Dateien:

**Befehlsauflistung 11:** Konfigurations Dateien für Rc-Scripte

```
/etc/conf.d/<name of rc-script>
/etc/conf.d/basic
/etc/rc.conf
```

**Notiz**
Diese drei werden in der aufgelisteten Reihenfolge eingelesen.
**Wichtig**
Alle **NET** Dienste lesen also /etc/conf.d/net aus.

# 10. Utilities und hilfreiche Scripts

## 10.1 Das rc-update Utility

Rc-update ist das Hauptwerkzeug, um Dienste zu einem bestimmten Runlevel zu ergänzen oder von einem solchen zu entfernen. Es wird darüber hinaus auch "depscan" aufrufen, um den Abhängigkeits Cache zu aktualisieren.

**Befehlsauflistung 12:** Metalog zum standard Runlevel ergänzen

```
# rc-update add metalog default
```

**Befehlsauflistung 13:** Metalog vom standard Runlevel entfernen

```
# rc-update del metalog default
```

**Notiz**
Um mehr Hilfe zu erhalten, rufen Sie einfach rc-update ohne weitere Argumente auf

## 10.2 Das depscan.sh Helfer Script

Um eine vollständige Dokumentation zu bieten, wird depscan.sh hier erwähnt. Es wird genutzt, um einen Abhängigkeits Cache zu erstellen, dieser ist eigentlich eine Aufzeichnung der Abhängigkeiten zwischen den Diensten.

Es sollte immer dann gestartet werden, wenn ein neues Rc-Script nach /etc/init.d/ ergänzt wird. Da jedoch rc-update dieses Script automatisch aufruft, wird der normale Benutzer normalerweise nicht mit depscan in Berührung kommen.