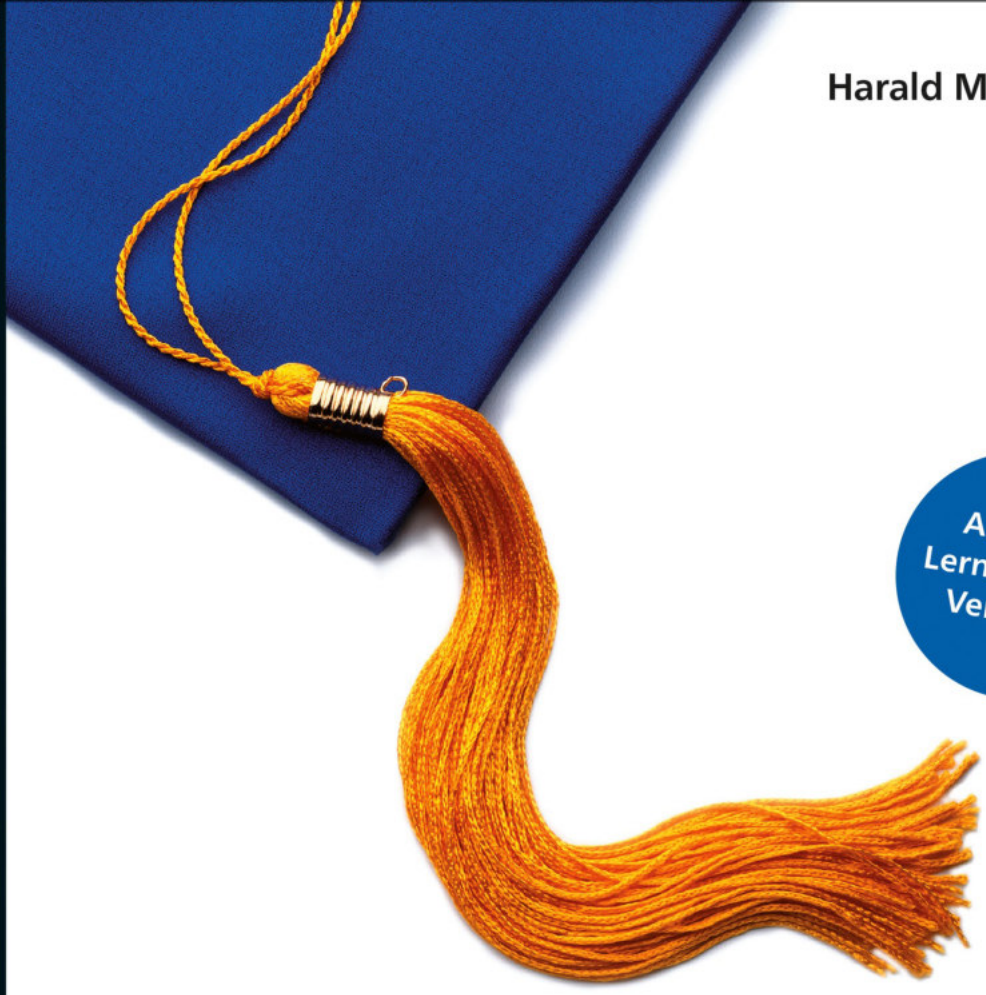
Harald Maaßen



Aktuelle Lernziele der Version 5

LPIC-1

Sicher zur erfolgreichen Linux-Zertifizierung

- Vorbereitung auf die Prüfungen 101 und 102
- Kommentierte Testfragen für beide Prüfungen
- Prüfungssimulator mit sofortiger Auswertung

Prüfungssimulator zum Download

Rheinwerk Computing

103 GNU- und Unix-Kommandos

Ein großer Teil dessen, was ganz selbstverständlich als Linux bezeichnet wird, entspringt in Wirklichkeit dem GNU-Projekt. In dem nun folgenden Kapitel geht es um einen kleinen Teil von GNU. GNU is Not Unix. GNU is Not Unix. GNU is Not U...

103.1 Auf der Kommandozeile arbeiten

Wichtung: 4

Beschreibung: Kandidaten sollten in der Lage sein, über die Kommandozeile mit Shells und Kommandos zu interagieren. Dieses Prüfungsziel setzt die Bash als Shell voraus.

Wichtigste Wissensgebiete:

- einzelne Shell-Kommandos und einzeilige Kommandofolgen verwenden, um einfache Aufgaben auf der Kommandozeile zu lösen
- die Shell-Umgebung verwenden und anpassen, etwa um Umgebungsvariablen zu definieren, zu verwenden und zu exportieren
- die command-history verwenden und ändern
- Kommandos innerhalb und außerhalb des definierten Suchpfads aufrufen

Liste wichtiger Dateien, Verzeichnisse und Anwendungen:

- `bash`
- `echo`
- `env`
- `export`
- `pwd`
- `set`
- `unset`
- `type`
- `which`
- `man`
- `uname`
- `history`
- *.bash_history*
- *Quoting*

Allgemeines

Inzwischen gibt es für Linux zwar etliche grafische Frontends und auch die meisten administrativen Tätigkeiten können inzwischen mit der Maus ausgeführt werden, aber das wichtigste Eingabegerät bleibt bei der Systemverwaltung immer noch die Tastatur. Damit Sie im Textmodus mit dem Computer kommunizieren können, benötigen Sie eine Shell. Der Begriff *Shell* bedeutet in der englischen Sprache *Schale*, was genau den Punkt trifft, weil man über diese Schale mit dem Kern(el) Informationen austauschen kann. Unter Linux wird standardmäßig die Bash als Shell verwendet. Hierbei handelt es sich um eine Weiterentwicklung der ursprünglichen Unix-Shell *sh*. Die Shell *sh* ist immer noch Bestandteil von Linux, wird aber zumindest als Login-Shell nicht mehr so oft verwendet. Die Shell *sh* wurde von Stephen Bourne bereits 1977 entwickelt, während die *Bash* (*Bourne Again Shell*) als Teil des GNU-Projektes erst Ende der 80er Jahre des letzten Jahrhunderts von Brian Fox und Chet Ramey geschrieben wurde. Die Bezeichnung Bash ist ein Wortspiel und kann als *Born Again* oder *Bourne Again* gelesen werden. Weitere bekannte, unter Linux verwendete Shells sind *ksh* (*Korn-Shell*), *csh* (*C-Shell*), *tcsh* (eine Erweiterung der *C-Shell*), *zsh* (mit *ksh* verwandte *Z-Shell*), *rsh* (*Remote Shell*) und einige mehr.

Aufbau eines Shell-Kommandos

Ein Kommando besteht aus bis zu drei verschiedenen Komponenten. Die erste Komponente ist das Kommando selbst, also eine ausführbare Datei, ein Skript oder ein Shell-interner Befehl. Ein Kommando kann auch für sich allein stehen und ohne Optionen oder Argumente funktionieren:

```
archangel:~ # mount
```

Wenn das Kommando `mount` ohne Optionen oder Argumente ausgeführt wird, zeigt es aktuell eingehängte Dateisysteme an. Es gibt aber auch Konstellationen, in denen ein Kommando mit einer oder mehreren Optionen ausgeführt wird:

```
archangel:~ # mount -a
```

In diesem Beispiel wurde `mount` mit der Option `-a` ausgeführt. Das Programm `mount` würde mit dieser Option alle in der Datei */etc/fstab* aufgeführten Dateisysteme einhängen.

Es ist aber auch möglich, `mount` ausschließlich mit Argumenten zu versehen:

```
archangel:~ # mount /dev/sda1 /boot
```

Argumente unterscheiden sich von Optionen. Eine Option sagt einem Programm, wie es sich verhalten soll. Argumente teilen einem Programm mit, was es verarbeiten

soll. Es ist möglich, einem Programm sowohl mehrere Optionen als auch mehrere Argumente zu übergeben. Bei vielen Programmen ist die Reihenfolge sogar variabel:

```
mount -o username=antje,password=xy //fs1/data /mnt/ -t smbfs
```

Das Beispiel zeigt eine Mischung mit einer Option vorn, einer Option hinten und den Argumenten in der Mitte. Der `mount`-Befehl baut in diesem Fall eine Netzwerkverbindung zu einem Windows-Computer auf. Konsultieren Sie im Zweifelsfall immer die Manpage eines Programms, wenn Sie herausfinden wollen, wie Optionen und Argumente übergeben werden müssen.

Übergabe der Optionen

Bei der Übergabe von Optionen gibt es mehrere gängige Methoden. Einige Kommandos verlangen, dass den Optionen ein Bindestrich vorangestellt wird, andere arbeiten auch ohne einleitenden Bindestrich. Außerdem sehen viele Programme vor, dass ganze Wörter als Optionen verwendet werden. Diesen werden in der Regel zwei Bindestriche vorangestellt. Das folgende Beispiel kennen Sie sinngemäß schon aus dem vorangegangenen Kapitel:

```
archangel:~ # rpm --install -vh /i586/fortune-1.0-860.i586.rpm
Preparing.. ########################################### [100 %]
```

Diesmal wurden absichtlich mehrere Formen der Optionsübergabe kombiniert. Es ist ein Optionswort mit zwei vorangestellten Bindestrichen und zwei Optionen mit einem gemeinsamen vorangestellten Bindestrich vorhanden. Ein weiterer prominenter Vertreter der flexiblen Programme ist `tar`. Sie können `tar` mit normalen Optionen oder Optionswörtern verwenden. Außerdem kann `tar` auch Optionen ohne vorangestellte Bindestriche übernehmen. Die folgenden vier Kommandos führen bei `tar` zum selben Ergebnis:

```
# tar -x -z -v -f xmbmon205.tar.gz
# tar -xzvf xmbmon205.tar.gz
# tar xvzf xmbmon205.tar.gz
# tar --extract --gzip --verbose --file=xmbmon205.tar.gz
```

Umgebungsvariablen und Shellvariablen

Grundsätzlich haben Variablen die Aufgabe, Werte aufzunehmen. Diese können von einem Programm ausgewertet und weiterverarbeitet werden. Variablen beeinflussen aber auch das Erscheinungsbild und Verhalten des ganzen Betriebssystems. Sie können sich den Inhalt einer Variablen einfach durch die Eingabe des Kommandos `echo`, gefolgt von einem Dollarzeichen und dann der Variablen selbst, ausgeben lassen:

```
archangel:~ # echo $OSTYPE
linux
```

Man unterscheidet auf einem Unix- oder auch Linux-System zwischen Umgebungsvariablen und Shellvariablen. Die Unterschiede zwischen diesen beiden Variablentypen liegen einerseits darin, durch welche Konfigurationsdateien sie deklariert werden und demzufolge wann und durch welches Programm sie ausgewertet werden, und andererseits in ihrem Wirkungsbereich:

- Umgebungsvariablen gelten für alle Shells, die ein Benutzer verwendet. Die Inhalte dieser Variablen werden an Subshells vererbt. Das bedeutet, dass beim Aufruf einer Subshell ein automatischer Export der Variablen in diese Subshell stattfindet. Bei Umgebungsvariablen werden normalerweise Großbuchstaben verwendet.
- Shellvariablen müssen in jeder Subshell, die durch den Benutzer oder ein Skript gestartet wird, neu deklariert werden. Es findet standardmäßig keine Vererbung statt. Für Shellvariablen verwendet man üblicherweise Kleinbuchstaben.

Wenn sich ein Benutzer an einem System anmeldet, werden bereits mehrere Variablen definiert. Wenn er dann weitere Shells öffnet oder Skripte ausführt, werden weitere Variablen deklariert. Dafür sind folgende Konfigurationsdateien zuständig:

- */etc/profile* ist die erste Konfigurationsdatei, die bei der Anmeldung eines Benutzers eingelesen wird. Sie enthält erste Umgebungsvariablen und (für die Prüfung besonders wichtig!) die erste PATH-Anweisung. Damit sich Änderungen an dieser Datei auswirken, muss sich der Benutzer gegebenenfalls ab- und wieder anmelden.
- */etc/bashrc* kann von der *.bashrc* eines Benutzers geladen werden. Sie enthält systemweite Einstellungen, Aliase und Funktionen. Diese Datei wird beim Start jeder Shell neu eingelesen und erfordert deshalb nach Änderung keine Neuanmeldung des Benutzers.
- *~/.bash_profile* wird (falls vorhanden) nur bei einer Neuanmeldung eingelesen und sofort nach */etc/profile* ausgeführt. Sie beinhaltet zusätzliche Pfadanweisungen (z. B. das Heimatverzeichnis), den zu verwendenden Standardeditor und benutzerspezifische Umgebungsvariablen.
- *~/.bash_login* ist eine Alternative zu *.bash_profile* und wird auch nur dann abgearbeitet, wenn die Datei *.bash_profile* nicht existiert. Auch diese Datei wird nur während der Anmeldung verwendet. Inhalt und Verwendungszweck entsprechen der *.bash_profile*.
- *~/.profile* ist die ursprüngliche Konfigurationsdatei der Bash. Sie wird nur während der Anmeldung eingelesen und auch nur dann, wenn weder eine *.bash_profile*- noch eine *.bash_login*-Datei im Verzeichnis des Benutzers existieren. Inhalt und Verwendungszweck entsprechen der *.bash_profile* und *.bash_login*.

- *~/.bashrc* ist die andere, ursprüngliche Konfigurationsdatei der Bash. Sie wird in jedem Fall eingelesen, und zwar auch beim Aufruf einer neuen Shell. Nach Änderungen in dieser Datei ist entsprechend keine Neuanmeldung des Benutzers erforderlich. Sie beinhaltet im Wesentlichen Aliase und Funktionen.
- *~/.bash_logout* ist eine optionale Datei, die ausgeführt wird, wenn der Benutzer sich abmeldet. Sie könnte z. B. den Monitor löschen.

Sie können sich alle aktuell gesetzten Shellvariablen anzeigen lassen, indem Sie das Kommando `set` ohne Optionen und Argumente verwenden. Da die Ausgabe des Kommandos recht umfangreich ist, lohnt sich eine Ausgabe mit dem Pager `less`:

```
archangel:~ # set | less
BASH=/bin/bash
BASH_ARGC=()
BASH_ARGV=()
... ca. 800 Zeilen wurden abgeschnitten ...
```

Um die Umgebungsvariablen aufzulisten, nutzen Sie das Kommando `env`. Auch hier sollten Sie `less` zur Anzeige verwenden:

```
archangel:~ # env | less
LESSKEY=/etc/lesskey.bin
HOSTNAME=archangel
HOST=archangel
TERM=xterm
SHELL=/bin/bash
... ca. 60 Zeilen wurden abgeschnitten ...
```

Wenn Sie selbst eine Variable deklarieren müssen, können Sie das einfach durch Eingabe der Variablen, gefolgt von dem zuzuweisenden Wert an der Eingabeaufforderung, erledigen. Variablen, die auf diese Art Werte erhalten, werden aber nicht automatisch in Subshells exportiert, wie die folgende Befehlsfolge demonstriert:

```
archangel:~ # x=5000
archangel:~ # echo $x
5000
archangel:~ # bash
archangel:~ # echo $x

archangel:~ #
```

Im ersten Schritt wurde der Variablen `x` der Wert `5000` zugewiesen. Anschließend wurde die Variable mit dem `echo`-Kommando ausgegeben. In der vierten Zeile wird mit dem Kommando `bash` eine Subshell gestartet. Der Versuch, die Variable mit dem

echo-Befehl auszugeben, schlägt fehl, weil keine Vererbung der Variablen in die Subshell stattgefunden hat.

Wenn der Inhalt einer Variablen in einer Subshell benötigt wird, muss die Variable exportiert werden. Beispiel:

```
archangel:~ # Name=Hans
archangel:~ # echo $Name
Hans
archangel:~ # export Name
archangel:~ # bash
archangel:~ # echo $Name
Hans
```

Diesmal wurde die Variable Name mit dem Kommando export exportiert, bevor die Subshell gestartet wurde. Wie der zweite echo-Befehl zeigt, steht die Variable danach in der Subshell zur Verfügung.

Es ist aber auch möglich, die Deklaration und den Export einer Variablen in einem einzigen Schritt durchzuführen. Das wird in der ersten Zeile des folgenden Beispiels demonstriert:

```
archangel:~ # export Name=Willi
archangel:~ # bash
archangel:~ # echo $Name
Willi
```

Prüfungstipp

Achtung: Eine Variable kann niemals in eine übergeordnete Shell exportiert werden, sondern nur in eine Subshell. In der Prüfung wird man Sie vielleicht mit diesem Verhalten konfrontieren.

Wenn Sie den Inhalt einer Variablen nicht mehr benötigen, sollten Sie diese mit dem Kommando unset wieder zurücksetzen. Es ist sinnvoll, zu Beginn eines Skripts zunächst alle verwendeten Variablen zurückzusetzen. Damit verhindern Sie, dass eventuell exportierte Variablen mit gleichen Bezeichnungen den Funktionsablauf Ihres Skripts stören. Beispiel:

```
archangel:~ # unset Name
```

Beliebte Variablen für Prüfungen

Manche Variablen werden in der Prüfung immer wieder abgefragt. Dazu gehören besonders die folgenden:

- $HISTSIZE definiert die Anzahl der Kommandos, die in der Befehls-History aufbewahrt werden. Diese Variable wird normalerweise in der Datei */etc/profile* festgelegt.
- $PS1 bestimmt das Aussehen der Eingabeaufforderung (Prompt). Beispiel:

  ```
  root@archangel:~# echo $PS1
  \u@\h:\w$
  ```

  Die Zeichen \u stehen für Username. Das @ wird normal ausgegeben. Die Zeichen \h werden mit dem Host-Namen ersetzt. Es folgt ein Doppelpunkt. Anschließend zeigt \w für Working Directory das aktuelle Verzeichnis an. Die Tilde ~ repräsentiert hierbei das Heimatverzeichnis des Benutzers. Es folgt ein >. Der Prompt kann durch Änderung der Variablen PS1 den eigenen Wünschen angepasst werden. Es empfiehlt sich dann eine Konfiguration in der Datei */etc/bashrc*.
- $? enthält das Errorlevel des zuletzt ausgeführten Kommandos. In der Regel bedeutet der Wert 0 in dieser Variablen, dass das letzte Programm erfolgreich ausgeführt wurde. Die Fehlerwerte variieren von Programm zu Programm. Beliebte Werte sind 1 und 127. Beispiel:

  ```
  archangel:~ # Kommando, das es nicht gibt
  bash: Kommando: command not found
  archangel:~ # echo $?
  127
  ```

  Das »Kommando, das es nicht gibt« hat ein Errorlevel von 127 zur Folge.
- $1, $2 usw. enthalten die Optionen und Argumente, die an ein Programm übergeben werden. Sie werden im Normalfall vom gestarteten Programm oder Skript selbst ausgewertet. Mehr über dieses Thema erfahren Sie im zweiten Teil dieses Buches.

Bash-Befehls-History und automatisches Vervollständigen von Befehlen

Um auf der Kommandozeile effizienter arbeiten zu können, stellt die Bash Ihnen noch zwei hervorragende Hilfsfunktionen zur Verfügung. Stellen Sie sich vor, Sie wollen das folgende Programm entpacken:

```
linux:/dl/ # tar xvzf ez-ipupdate-3.0.11b7-linux-i386.tar.gz
```

Die Wahrscheinlichkeit ist recht hoch, dass man sich bei diesem langen Dateinamen vertippt, und Dateinamen von solchem Ausmaß sind unter Linux nicht gerade die Ausnahme. Mit der automatischen Befehlsvervollständigung der Bash ist das kein Problem mehr. Der Befehl muss nur so weit ausgeschrieben werden, dass die Shell diesen eindeutig identifizieren kann. Danach betätigen Sie die Tabulatortaste ⇆:

```
linux:/dl/ # tar xvzf ez[TAB]
```

Dieser Befehl sollte ausreichen, um den Tar-Ball zu entpacken. Wenn es beim Betätigen der Tabulatortaste mehrere Möglichkeiten zur Vervollständigung des Kommandos gibt, erhalten Sie einen akustischen Hinweis. Sie können dann durch wiederholtes Betätigen der Tabulatortaste eine Auswahl der Möglichkeiten zur Vervollständigung erhalten.

Eine weitere nützliche Funktion der Bash ist die Befehls-History. Alle Befehle, die Sie verwenden, werden in einer Datei mit der Bezeichnung *.bash_history* abgespeichert. Der Punkt zu Beginn der Datei ist schon ein Hinweis darauf, dass es sich um eine Datei handelt, die im Heimatverzeichnis eines Benutzers abgelegt wird. Wie viele Kommandos in der History-Liste zwischengespeichert werden, wird über die Variable `HISTSIZE` festgelegt. Sie finden diese Variable normalerweise in der Datei */etc/profile*.

Um bereits verwendete Befehle zu wiederholen und gegebenenfalls zu editieren, stehen Ihnen verschiedene Möglichkeiten zur Verfügung. Die beliebteste ist wahrscheinlich die Verwendung der (Cursor nach oben)-Taste. Hierdurch werden die zuletzt verwendeten Befehle in umgekehrter Reihenfolge aufgerufen. Dieses Verhalten kennen alte DOS-Anwender auch von *doskey*. Weitere Möglichkeiten zur Verwendung der History sind:

- `!!` – Dieses Kommando wird als Bang-Bang bezeichnet und führt den letzten Befehl der History noch einmal aus.
- `!n` – Wenn Sie einfach den Befehl `history` eingeben, erhalten Sie eine nummerierte Auflistung der zuletzt verwendeten Befehle. Sie können dann `n` mit der Nummer des gewünschten Befehls ersetzen.
- `!-n` führt den letzten Befehl `-n` aus. Wenn Sie `!-2` eingeben, wird der vorletzte Befehl wiederholt.
- `!<Zeichenkette>` führt den letzten Befehl aus, der mit `<Zeichenkette>` beginnt. Wenn Sie etwa das Kommando `tail /var/log/syslog` noch einmal wiederholen möchten, geben Sie einfach `!ta` ein.
- `!?<Zeichenkette>` führt den letzten Befehl aus, in dem `<Zeichenkette>` vorkommt.

PATH-Variable

Wenn Sie ein Programm ausführen wollen, das sich nicht in einem in Ihrer PATH-Variablen enthaltenen Verzeichnis befindet, müssen Sie den kompletten Pfad zu diesem Programm auf der Kommandozeile mit angeben.

Wenn Sie ein Programm ohne Angabe eines Pfades starten, prüft die Shell zunächst, ob es sich um ein internes Kommando der Shell handelt. Dazu zählen: `echo`, `bg`, `fg`, `jobs`, `kill`, `pwd`, `set`, `unset` und viele mehr.

Falls kein passendes Kommando gefunden wird, sucht die Shell das Programm in allen Verzeichnissen, die in der PATH-Variablen enthalten sind. Sollten mehrere Programme mit dem gleichen Namen vorhanden sein, wird das zuerst gefundene ausgeführt. Sie können in einem solchen Fall das Kommando `which` verwenden, um festzustellen, welches Programm die Shell automatisch ausführt:

```
archangel:~ # which less
/usr/bin/less
```

Es ist zu beachten, dass ein Programm auch dann nicht ohne Pfadangabe ausgeführt wird, wenn man sich aktuell in dessen Verzeichnis befindet. Dieses Verhalten gibt es unter DOS, aber nicht bei Linux. Sie können ein Programm, das sich im aktuellen Verzeichnis befindet, aber auch starten, indem Sie *./* dem Programm voranstellen. Die Zeichen *./* repräsentieren das aktuelle Verzeichnis. Sie können z. B. Konfigurationsskripte von Tar-Balls im aktuellen Verzeichnis ausführen, indem Sie folgende Syntax verwenden:

```
archangel:/usr/src/xmbmon205 # ./configure
```

Wenn Sie einen Prompt verwenden, der Ihnen nicht anzeigt, in welchem Verzeichnis Sie sich gerade befinden, verwenden Sie das Kommando `pwd` (*Path of Working Directory*) ohne Optionen. Der Pfad wird Ihnen dann angezeigt:

```
archangel:/usr/src/xmbmon205 # pwd
/usr/src/xmbmon205
```

Sie können sich den Inhalt Ihrer PATH-Variablen, genau wie den anderer Variablen, anzeigen lassen:

```
archangel:/ # echo $PATH
/sbin:/usr/sbin:/usr/local/sbin:/root/bin:/usr/local/bin:/usr/bin:/usr/X11R6/
bin:/bin:/usr/games:/opt/gnome/bin:/opt/kde3/bin:/usr/lib/jvm/jre/bin
```

Quoting

Wenn in Argumenten bei der Befehlsausführung Leerzeichen oder andere Zeichen, die durch die Shell interpretiert würden, vorkommen, müssen diese Zeichen geschützt werden. Man sagt dann, dass das Zeichen *maskiert* oder im Jargon *escaped* wird. Der Begriff *Quoting* leitet sich davon ab, dass für diesen Vorgang häufig Hochkommata, also *Quotes* verwendet werden.

In der Praxis werden Quotes insbesondere zum Schutz regulärer Ausdrücke verwendet. Quotes sind:

- \ (Backslash) schützt nur das direkt im Anschluss folgende Zeichen vor der Expansion durch die Shell.
- ' (einfache Hochkommata) schützen alle eingeschlossenen Zeichen inklusive Variablen vor der Expansion.
- " (doppelte Hochkommata) schützen eingeschlossene Zeichen vor der Expansion, aber keine Variablen und Historyexpandierungen.

Hinweis

Verwechseln Sie bitte nicht die Verwendung von Hochkommata und Akzentzeichen. Insbesondere ein ` (Backquote, Backtick) hat in der Shell eine völlig andere Funktion.

Das Kommando uname

Der Befehl `uname` gibt Informationen zum laufenden System aus. Mit entsprechenden Optionen versehen, können Sie hier auch die Versionsnummer des laufenden Kernels überprüfen.

Zum Beispiel:

```
root@archangel:~# uname -r
2.6.24-23-generic
```

oder:

```
root@archangel:~# uname -a
Linux archangel 2.6.24-23-generic #
1 SMP Mon Jan 26 00:13:11 UTC 2009 i686 GNU/Linux
```

Die von `uname` generierten Informationen können Sie im Folgenden benutzen, um sich genauere Informationen über bestimmte Module anzusehen oder um die zum laufenden Kernel passenden Module zu laden bzw. zu entladen.

Manpages

Manpages sind ein komfortabel zu benutzendes Hilfesystem und stehen für fast alle Linux-Befehle zur Verfügung. Geben Sie einfach den Befehl `man`, gefolgt von dem zu erklärenden Befehl, auf der Kommandozeile ein, und schon erhalten Sie eine genaue Beschreibung.

Funktionsweise der Manpages

Immer wenn Sie eine Manpage aufrufen, konsultiert das Programm `man` die in der Variablen `$MANPATH` enthaltenen Verzeichnisse. Hier können mehrere Verzeichnisse

auf einmal hinterlegt sein. In nach Sektionen unterteilten Unterverzeichnissen befinden sich die Dateien der Manpages im komprimierten Zustand. Das Mansystem dekomprimiert die angeforderte Datei und übergibt sie anschließend an einen Pager. Dieser zeigt daraufhin die gewünschte Manpage an.

$MANPATH und die Datei manpath.config

Wie bereits erwähnt, enthält die Variable `$MANPATH` Informationen darüber, wo sich die Manpages überhaupt befinden. Genauere Informationen darüber erhalten Sie in der Konfigurationsdatei */etc/manpath.config*. Wenn dem System neue Manpages hinzugefügt werden, wird diese Datei normalerweise bei Bedarf auch automatisch aktualisiert. Ein manueller Eingriff ist in der Regel nicht nötig. Um den aktuellen MANPATH zu ermitteln, geben Sie einfach das Kommando `manpath` ohne Parameter ein. Das Ergebnis sieht dann beispielsweise so aus:

```
archangel:/ # manpath
/usr/share/man:/usr/local/man:/usr/X11R6/man:/opt/gnome/share/man
```

Ein weiterer wichtiger Pfad ist der, in dem das Mansystem seine Temporärdateien ablegt. Wie bereits erwähnt, sind Manpages im Rohformat komprimiert. Deshalb werden sie vor dem Anzeigen in ein temporäres Verzeichnis dekomprimiert. In der Regel handelt es sich hierbei um das Verzeichnis */var/cache/man*. Dieses Verzeichnis wird, gestützt durch den Daemon *cron*, zyklisch von *catman* bereinigt. Um das Cache-Verzeichnis Ihres Computers ausfindig zu machen, geben Sie das Kommando `manpath -c` ein:

```
archangel:/ # manpath -c
/var/cache/man:/var/cache/man/local:/var/cache/man/opt
```

Die Sektionen des Mansystems

Das Mansystem ist in mehrere Sektionen unterteilt. Diese Sektionen sind folgendermaßen gegliedert:

- 1 – ausführbare Programme für Benutzer
- 2 – Systemaufrufe (Funktionen, die durch den Kernel unterstützt werden)
- 3 – Bibliothekaufrufe (Libraries)
- 4 – besondere Dateien (normalerweise in */dev*)
- 5 – Dateiformate und Konventionen
- 6 – Spiele
- 7 – Sonstiges (Makro-Pakete und Konventionen)
- 8 – administrative Programme (nur für den User root)
- 9 – Kernel-Routinen (nicht standardisiert)

Zu vielen Programmen oder Dateien gibt es überhaupt nur eine einzige Manpage. Sollte es aber mehrere geben, so wird jene angezeigt, die numerisch betrachtet in der niedrigsten Sektion steht. Ein Beispiel hierfür ist *crontab*. Für *crontab* gibt es zwei Einträge in den Manpages, weil es sich bei *crontab* sowohl um ein ausführbares Programm als auch um eine Konfigurationsdatei handelt. Deshalb gibt es für *crontab* einen Eintrag unter Sektion 1 und einen unter Sektion 5. Brauchen Sie nun Informationen über die Konfigurationsdatei *crontab*, so können Sie diese mit folgendem Kommando aufrufen:

```
man 5 crontab
```

Hilfe zum Programm `crontab` erhalten Sie, indem Sie die Manpage ohne Angabe der Sektion aufrufen:

```
man crontab
```

Aufbau von Manpages

Der Aufbau einer Manpage ist festgelegt und sollte immer die folgenden Elemente enthalten:

- *Name* – die Bezeichnung des Elements mit einer kurzen Beschreibung
- *Synopsis* – eine vollständige Kurzbeschreibung der Syntax
- *Description* – eine ausführliche Beschreibung des Elements
- *Defaults* – voreingestellte Parameter
- *Overview* – ein Überblick über die komplexeren Zusammenhänge
- *Options* – Optionen und deren Beschreibung
- *Return Values* – Informationen über eventuelle Rückgabewerte, teilweise auch als Exit-Status bezeichnet
- *See also* – Verweise auf artverwandte Themen
- *Bugs* – bekannte Fehler
- *Files* – Konfigurationsdateien u. Ä.

Es sind noch mehr Elemente denkbar, aber die hier aufgeführten sind die wohl geläufigsten.

Verwandte Befehle

Es gibt noch einige Befehle, die mit `man` verwandt sind bzw. die man zumindest thematisch in diesem Zusammenhang erwähnen muss, weil sie auch in der Prüfung in einen gemeinsamen Kontext fallen. Vorrangig sind hier die Kommandos `whatis` und `apropos` zu nennen. Der Grund hierfür ist, dass diese beiden auch auf die Dateien des Mansystems zugreifen. Der Befehl `whatis` durchsucht hierfür nur das Feld *Name* aller

verfügbaren Manpages. Das ist besonders dann praktisch, wenn man damit rechnen muss, dass es zu einem Element in mehreren Sektionen Einträge gibt, z. B. in `crontab`:

```
archangel:/ # whatis crontab
crontab (1)      - maintain crontab files for individual users
crontab (5)      - tables for driving cron
crontab (1p)     - schedule periodic background work
```

Wenn Sie vergessen sollten, `whatis` einen Abfragewert als Argument zu übergeben, fragt das Programm übrigens noch einmal nach:

```
archangel:/ # whatis
whatis what?
```

Das Programm `apropos` ist ebenfalls nur ein verkürzter Spezialfall der Manpages. Im Gegensatz zu `whatis` durchsucht `apropos` nicht nur das Feld *Name*, sondern außerdem das Feld *Description*. Deshalb hat man die Möglichkeit, mit `apropos` nach einem Programm zu suchen, das eine bestimmte Aufgabe erfüllt. Stellen Sie sich vor, Sie besitzen ein neues Bandlaufwerk, haben aber keine Idee, welche Programme ein solches ansteuern können. Geben Sie einfach folgendes Kommando ein:

```
archangel:~ # apropos tape
mt (1)              - control magnetic tape drive operation
st (4)              - SCSI tape device
smbtar (1)          - shell script for backing up SMB/CIFS
```

Immerhin ergab die Abfrage drei Treffer, mit denen sich schon etwas anfangen lässt. Wenn Sie die hiermit korrespondierenden Manpages gelesen haben und eventuellen *See also*-Hinweisen am Ende der jeweiligen Page gefolgt sind, dürften schon allein dadurch viele Fragen zum Thema Streamer geklärt worden sein.

Es folgen nun einige nützliche Kommandos, die sowohl für die Praxis als auch für die Prüfung interessant sind, die aber nicht direkt mit dem Mansystem in Verbindung stehen.

`whereis` zeigt an, wo sich ein Programm, seine Konfigurationsdatei(en) und die zugehörige(n) Manpage(s) befinden. Um dies herauszufinden, stellen Sie die Frage: Wo ist `cron`?

```
archangel:~ # whereis cron
cron: /usr/sbin/cron /etc/cron.d /etc/cron.daily /etc/cron.hourly
/etc/cron.monthly /etc/cron.weekly /usr/lib/cron /usr/share/man/man8/
cron.8.gz
```

`which` zeigt den vollständigen Pfad eines Kommandos an. Das ist nützlich, wenn Sie mehrere Versionen eines Programms besitzen und wissen möchten, welche von diesen bei einer Kommandoeingabe ohne Pfadhinweis ausgeführt wird:

```
archangel:~ # which less
/usr/bin/less
```

`which` zeigt hier an, in welchem Pfad sich das Programm `less` befindet. Wenn ein Alias zu einem Programm existiert, wird auch dieser angezeigt:

```
archangel:~ # which ls
alias ls='ls --color=auto'
        /usr/bin/ls
```

Die Kommandos `set` und `unset` werden von `which` nicht gefunden, weil es sich um interne Kommandos der Shell handelt:

```
archangel:~ # which unset
/usr/bin/which: no unset in (/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/
bin:/root/bin)
```

Sie können aber für eine entsprechende Überprüfung den internen Befehl `type` verwenden:

```
archangel:~ # type unset
unset is a shell builtin
```

Bei Programmen, die im Suchpfad liegen, macht `type` eine ähnliche Ausgabe wie `which`:

```
archangel:~ # type top
top ist /usr/bin/top
```

[+]

Prüfungstipp

Bei der Prüfung sollten Sie mit der Verwendung des Mansystems bestens vertraut sein.

103.2 Textströme mit Filtern verarbeiten

Wichtung: 2

Beschreibung: Kandidaten sollten in der Lage sein, Filter auf Textströme anzuwenden.

Wichtigste Wissensgebiete:

Textdateien und Ausgabeströme durch Textfilter schicken, um die Ausgabe mit Standard-UNIX-Kommandos aus dem GNU-textutils-Paket zu verändern

Liste wichtiger Dateien, Verzeichnisse und Anwendungen:

- `bzcat`
- `cat`
- `cut`
- `head`
- `less`
- `md5sum`
- `nl`
- `od`
- `paste`
- `sed`
- `sha256sum`
- `sha512sum`
- `sort`
- `split`
- `tail`
- `tr`
- `uniq`
- `wc`
- `xzcat`
- `zcat`

Allgemeines

Unter Linux stehen Ihnen viele Programme zur Verfügung, mit denen Sie Texte bzw. Textströme manipulieren können. Bei einigen dieser Programme handelt es sich auch um Textfilter. In den folgenden Abschnitten werden Sie einige dieser nützlichen Tools kennenlernen.

cat

Das Programm `cat` wird hauptsächlich benutzt, um Textdateien auf dem Bildschirm auszugeben oder in ein Programm umzuleiten. Ursprünglich steht `cat` für *concatenate* (verketten), und das Programm ist dazu gedacht, mehrere Dateien zu einer einzigen Datei zusammenzufassen. Sie können mit den entsprechenden Schaltern auch nicht druckbare Zeichen ausgeben. Im nächsten Beispiel wurde eine Datei mit folgendem Inhalt verwendet:

```
1               Birnen
2               Aepfel
3               Bananen
4               Erdbeeren
```

Die Abstände zwischen den Ziffern und dem Obst sind in Wirklichkeit je zwei Tabstopps. Mit `cat -A` können diese dargestellt werden, wie die folgende Ausgabe demonstriert:

```
archangel:/textfiles # cat textfile1 -A
1^I^IBirnen$
2^I^IAepfel$
3^I^IBananen$
4^I^IErdbeeren$
```

bzcat, xzcat und zcat

Bei den ersten beiden dieser Kommandos handelt es sich um Softlinks, die mit Dateikompressionsprogrammen verknüpft sind. Nämlich `bzcat` mit `bzip2` und `xzcat` mit `xz`.

`zcat` ist hingegen ein eigenständiges Programm, das aber in Zusammenhang mit `gzip` für die gleichen Aufgaben genutzt wird wie die beiden zuerst genannten Befehle. Alle drei Kommandos geben den Inhalt komprimierter Dateien in der Standardausgabe aus. Sie verhalten sich alle ähnlich wie `cat`, nur eben auf komprimierte Dateien bezogen. In Topic 103.3 finden Sie weitere Informationen zu diesem Thema.

head

Das Kommando `head` zeigt in der Voreinstellung die ersten zehn Zeilen einer Datei an. Es können auch mehrere Dateien angegeben werden. Der Ausgabe jeder einzelnen Datei wird dann der Dateiname vorangestellt:

```
archangel:/textfiles # head textfile1 textfile2
==> textfile1 <==
1               Birnen
2               Aepfel
3               Bananen
4               Erdbeeren
==> textfile2 <==
hellgruen
rot
gelb
dunkelgruen
```


Mit der Option `-n` kann die Anzahl der auszugebenden Zeilen auf einen anderen Wert als 10 eingestellt werden:

```
archangel:/textfiles # head -n 1 textfile1
1               Birnen
```

tail

In der Praxis erheblich wichtiger als `head` ist das Kommando `tail`. *Tail* bedeutet Ende, Heck oder auch Schwanz. Per Default gibt `tail` die letzten zehn Zeilen einer Textdatei aus. Das ist besonders nützlich, um schnell einen Blick auf das Ende einer Logdatei werfen zu können. Sowohl in der Praxis als auch in der Prüfung besonders beliebt ist die Ausgabe des *Syslogs* mit `tail`:

```
archangel:/ # tail /var/log/syslog
```

Genauso wie bei `head` können Sie mit `-n` die Anzahl der von `tail` auszugebenden Zeilen festlegen. Viel interessanter ist aber, dass `tail` eine Datei fortlaufend anzeigen kann, während sie von einem anderen Programm beschrieben wird. Das ist ebenfalls besonders praktisch, um den *Syslog* (diesmal in Echtzeit) zu beobachten. Sie beenden die Ausgabe dann mit Strg + C. Starten Sie Ihre Beobachtung mit:

```
archangel:/ # tail /var/log/syslog –f
```

Je nach verwendeter Distribution protokolliert Ihr System mittels `systemd-journald`. Es existieren dann unter Umständen keine klassischen Logfiles. Sie müssen dann andere Dateien für diesen Test heranziehen.

less

Bei dem Programm `less` handelt es sich um einen Pager. Das ist ein Programm, mit dem Textdateien angezeigt werden können. In den Anfangszeiten von Unix hat man hierfür das Programm `more` verwendet. `less` bietet allerdings etliche Vorteile gegenüber `more`. Ein Nachteil von `more` ist, dass `more` immer die gesamte Datei einliest, bevor es mit der Ausgabe auf dem Bildschirm beginnt. `less` liest immer nur so viele Daten ein, wie auch ausgegeben werden können, was bei großen Dateien, wie etwas Logfiles, sehr angenehm ist.

Eine lästige Einschränkung von `more` ist auch, dass man innerhalb einer Datei nur nach unten scrollen kann. Will man also Text anzeigen, der weiter oben im Dokument steht, muss `more` neu aufgerufen werden. Weiterhin bietet `less` eine Suchfunktion.

nl

Diese Abkürzung steht für *number lines*. Das Programm nummeriert also Zeilen. Damit die zweite Beispieltextdatei auch eine Nummerierung erhält, wird folgendes Kommando verwendet:

```
archangel:/textfiles # nl textfile2 > textfile3
archangel:/textfiles # cat textfile3
     1  hellgruen
     2  rot
     3  gelb
     4  dunkelgruen
```

Das Kommando `nl` versteht viele Optionen. Die Kenntnis dieser Optionen ist für die Prüfung nicht erforderlich.

Prüfungstipp

Achten Sie in der Prüfung bitte darauf, die Programme `nl` und `ln` nicht miteinander zu verwechseln.

wc

Das Programm `wc` (word count) zählt die Anzahl der Zeilen, Wörter und Bytes in einer Datei. In der Standardeinstellung zählt `wc` alle drei Werte in der oben angegebenen Reihenfolge. Sie können auch hier mehrere Dateien angeben oder Wildcards verwenden. Es werden dann alle Dateien einzeln bearbeitet, und zum Schluss wird die Summe gebildet. Beispiel:

```
archangel:/textfiles # wc *
  4   8  44 textfile1
  4   4  31 textfile2
  4   8  59 textfile3
 12  20 134 total
```

Innerhalb eines Skripts ist es oft nötig, nur einen einzigen dieser Werte für eine Datei zu erhalten. Sie können dann als Option angeben, welchen Wert Sie benötigen:

- `-c` zeigt nur die Anzahl der Bytes an.
- `-l` zeigt nur die Anzahl der Zeilen (lines) an.
- `-w` zeigt nur die Anzahl der Wörter an.
- `-m` zeigt nur die Anzahl der Zeichen an.

hexdump

Auch wenn der Name etwas anderes aussagt: Mit `hexdump` können Dateien hexadezimal, dezimal, im ASCII-Format oder als Oktaldump dargestellt werden. Das Programm kann im Gegensatz zu einem normalen Editor auch ohne Probleme Binärdateien einlesen. Eigentlich eignet sich mehr oder weniger alles zur Eingabe. Mit dem folgenden Beispiel wird die Partitionstabelle eines Computers angezeigt:

```
archangel:/boot # hexdump /dev/sda | head -n 32 | tail -n 5
00001b0 0000 0000 0000 0000 133b b839 0000 0180
00001c0 0001 fe83 063f 003f 0000 b708 0001 0000
00001d0 0701 fe83 ffff b747 0001 d8da 024a fe00
00001e0 ffff fe82 ffff 9021 024c 0ade 0017 fe00
00001f0 ffff fe83 ffff 9aff 0263 49c2 06ed aa55
```

Hier liest `hexdump` direkt von der Festplatte. Es wird mit dem Sektor 0 begonnen. Das `head`-Kommando sorgt dafür, dass nur die ersten 32 Zeilen gelesen werden. Jede Zeile enthält 16 Byte. Es werden also 512 Byte eingelesen, was der Größe des Master Boot Records (*MBR*) genau entspricht. Zum Schluss wird mit `tail` dafür gesorgt, dass nur die letzten fünf Zeilen des *MBR* ausgegeben werden. Darin ist die Partitionstabelle (im Beispiel fett gedruckt) enthalten.

od

Das Kommando `od` (octal dump) ähnelt vom Verwendungszweck her dem `hexdump`. Es hat allerdings einen kleineren Funktionsumfang und verwendet per Voreinstellung das Oktalformat. Mit der Option `-x` kann die Ausgabe auf hexadezimal umgestellt werden.

sort

Mit `sort` können die Zeilen von Textdateien sortiert werden. Es ist aber genauso gut möglich, andere Eingabequellen zu verwenden. Damit Sie eine sauber sortierte Ausgabe des `mount`-Kommandos erhalten, können Sie folgenden Befehl verwenden:

```
archangel:~ # mount | sort
/dev/fd0 on /media/floppy type subfs (rw,nodev,sync,procuid)
/dev/sda1 on /boot type ext2 (rw,acl,user_xattr)
/dev/sda2 on / type ext3 (rw,acl,user_xattr)
/dev/sda4 on /storage type reiserfs (rw)
/dev/sda1 on /media/512MB type subfs (rw, nodev,sync, procuid)
/dev/sdb1 on /media/WD_Passport type ext3 (rw)
... weitere Zeilen wurden abgeschnitten ...
```

Wichtige Optionen für `sort` sind:

- `-n` sortiert nach numerischen Kriterien.
- `-o` sorgt für die Umleitung der Ausgabe in eine Datei (outfile). Normalerweise gibt `sort` das Ergebnis an *stdout* aus.
- `-r` gibt das Ergebnis in umgekehrter Reihenfolge (reverse) aus.

Im folgenden Beispiel wird die Datei *textfile1* eingelesen und in numerisch umgekehrter Reihenfolge in die Datei *reversefile* geschrieben:

```
archangel:/textfiles # sort textfile1 -n -r -o reversefile
```

uniq

Wenn Sie eine Datei, die sich wiederholende Zeilen enthält, kürzen wollen (z. B. eine Logdatei), können Sie das Kommando `uniq` verwenden. Die Voraussetzung ist allerdings, dass diese Zeilen aufeinander folgen. Sollte es notwendig sein, auch die Duplikate zu entfernen, die nicht hintereinander liegen, schalten Sie einfach das Kommando `sort` vor.

split

Um große Dateien in mehrere kleinere Dateien zu unterteilen, können Sie das Programm `split` verwenden. Als Disketten noch ein gängiges Speichermedium waren, war diese Vorgehensweise üblich, um große Dateien auf mehrere Disketten zu verteilen. Auch die Übermittlung großer Datenmengen per Mail war zu Zeiten geringer Bandbreiten im Internet ein Grund, Dateien zu teilen. Eigentlich arbeitet `split` per Voreinstellung zeilenorientiert. Wenn keine entsprechenden Optionen vergeben werden, teilt `split` eine Datei in kleinere Dateien zu je 1.000 Zeilen. Der Sachverhalt wird im Beispiel deutlich:

Die folgende Datei ist 101 MB groß und soll per Mail versendet werden. Leider lässt der Internetprovider des Empfängers keine Mails zu, die größer als 12 MB sind:

```
-rwxr--r--   1 root root 101M Nov 24 14:09 grosses-paket
```

Um eine noch kleinere Toleranz zu gewährleisten, wird die Datei mit dem folgenden Kommando in nur 11 MB große Stücke geteilt:

```
archangel:/files # split -b 11m grosses-paket grosses-paket_
```

Die Option `-b` sorgt dafür, dass `split` nicht in Zeilen rechnet, sondern in Bytes. Mit `11m` wird die Größe der Teilpakete dann auf 11 MB festgelegt. Das letzte Argument `grosses-paket_` gibt das Präfix für die entstehenden Pakete an. Die Ausgabe des folgenden Kommandos spricht für sich selbst:


```
archangel:/files # ls -lha
total 203M
drwxr-xr-x   2 root root 4.0K Nov 24 14:15 .
drwxr-xr-x  25 root root 4.0K Nov 24 14:08 ..
-rwxr--r--   1 root root 101M Nov 24 14:09 grosses-paket
-rw-r--r--   1 root root  11M Nov 24 14:15 grosses-paket_aa
-rw-r--r--   1 root root  11M Nov 24 14:15 grosses-paket_ab
... einige Zeilen wurden herausgeschnitten ...
-rw-r--r--   1 root root  11M Nov 24 14:15 grosses-paket_ai
-rw-r--r--   1 root root 2.0M Nov 24 14:15 grosses-paket_aj
```

Damit der Empfänger die Dateifragmente wieder zusammensetzen kann, muss er lediglich das Programm `cat` in seinem eigentlichen Sinn (*concatenate*) verwenden:

```
archangel:/files # cat grosses-paket_a* >grosses-paket
```

Dieses Verfahren mit einfachen Bordmitteln funktioniert übrigens hervorragend.

cut, paste

Die beiden Programme `cut` und `paste` werden oft miteinander kombiniert oder nacheinander angewendet. Für die Beispiele kommen die folgenden beiden Dateien zum Einsatz:

```
archangel:/textfiles # cat textfile1
1:Birnen
2:Aepfel
3:Bananen
4:Erdbeeren
```

und

```
archangel:/textfiles # cat textfile2
1:hellgruen
2:dunkelgruen
3:gelb
4:rot
```

cut

Das Kommando `cut` (schneiden) kann Spalten einer Datei ausschneiden. Das bedeutet aber nicht, dass diese Spalten hinterher nicht mehr vorhanden sind. Im Gegenteil: Das Programm gibt die ausgeschnittenen Spalten nach *stdout* aus. Um die Spalten voneinander zu unterscheiden, muss `cut` wissen, welches Zeichen als Trenner (Delimiter) zwischen den Spalten verwendet wird. Dieses Zeichen ist in den aufgeführten

Beispielen ein Doppelpunkt. Die folgende Befehlszeile schneidet die zweite Spalte (`-f2` steht für field 2) aus der Datei *textfile1* aus. Als Delimiter wird ein Doppelpunkt verwendet (`-d:`):

```
archangel:/textfiles # cut -d: -f2 textfile1
Birnen
Aepfel
Bananen
Erdbeeren
```

Normalerweise würde man die Ausgabe des Befehls mit einem Redirektor in eine Datei umleiten. Es lohnt sich aber während der Konstruktion einer Befehlszeile immer, das Ergebnis vorläufig auf den Bildschirm zu geben.

paste

Mit `paste` können Sie zwei oder mehr Dateien zusammenführen. Im Gegensatz zur Arbeitsweise von `cat` werden die Dateien aber, vereinfacht ausgedrückt, nicht untereinander, sondern nebeneinander zusammengefügt. So können korrespondierende Zeilen von textbasierten Datenbanken miteinander verknüpft werden. Beispiel:

```
archangel:/textfiles # paste textfile1 textfile2
1:Birnen        1:hellgruen
2:Aepfel        2:dunkelgruen
3:Bananen       3:gelb
4:Erdbeeren     4:rot
```

Etwas unschön an dieser Ausgabe ist die doppelte Verwendung der Nummerierung. Aber auch für dieses Problem stehen Ihnen viele Bordmittel zur Verfügung, wie die folgenden Abschnitte zeigen.

sed

Der Streamingeditor `sed` ist zwar Bestandteil dieses LPI-Topics, wird aber aus praktischen Gründen erst in Topic 103.7 besprochen.

tr

Wenn einzelne Zeichen einer Textdatei durch andere ersetzt oder ganz gelöscht werden sollen, kommt das Programm `tr` (translate) zum Einsatz. Es bietet keine Möglichkeit, Dateinamen als Argumente zu übergeben, weshalb eine Übergabe der Datei(en) mittels `cat`, gefolgt von einem Redirektor, üblich ist. Komplexere Änderungen sollten Sie aber dem Stream-Editor `sed` überlassen. Da `sed` noch an anderer Stelle detailliert behandelt wird, soll er hier nicht weiter thematisiert werden. Im folgenden

Beispiel wird in einer Datei der Buchstabe *n* durch den Buchstaben *x* ersetzt. Natürlich ist das völlig sinnlos, zeigt aber sehr deutlich, wie `tr` arbeitet.

```
archangel:/textfiles # cat textfile1 | tr n x
1:Birxex
2:Aepfel
3:Baxaxex
4:Erdbeerex
```

Wichtige Optionen für `tr` sind:

- `-d` bzw. `--delete` löscht das angegebene Zeichen.
- `-c` bzw. `--complement` kehrt die Ausgabe ins Gegenteil.
- `-s` bzw. `--squeeze-repeats` unterdrückt sich wiederholende Zeichen.

Ein beliebtes Beispiel für die Verwendung von `tr` ist auch die Übersetzung aller Zeichen von *a bis z* in Großbuchstaben:

```
archangel:/textfiles # cat textfile1 | tr a-z A-Z
1:BIRNEN
2:AEPFEL
3:BANANEN
4:ERDBEEREN
```

Dasselbe Ergebnis können Sie aber auch mit folgendem Kommando erreichen:

```
archangel:/textfiles # cat textfile1 | tr [:lower:] [:upper:]
```

Diese Form der Argumente erlaubt eine Menge interessanter Konstellationen:

- `[:alnum:]` steht für alle Buchstaben und Ziffern.
- `[:alpha:]` repräsentiert alle Buchstaben.
- `[:blank:]` steht für Spaces und Tabstopps.
- `[:cntrl:]` symbolisiert alle Steuerzeichen.
- `[:graph:]` steht für alle druckbaren Zeichen (ohne Space).
- `[:upper:]` vertritt Großbuchstaben.
- `[:lower:]` vertritt Kleinbuchstaben.
- `[:print:]` bedeutet druckbare Zeichen (inklusive Space).
- `[:punct:]` repräsentiert alle Satzzeichen.
- `[:space:]` steht für alle White-Spaces (Space, Tabstopps, CR und LF).

Wenn Sie einzelne Steuerzeichen ersetzen wollen, muss ein Backslash vorangestellt werden. Das gilt insbesondere, wenn Sie den Backslash selbst ersetzen oder entfernen müssen.

- \NNN bezeichnet ein ASCII-Zeichen in Oktalschreibweise.
- \\ schützt den Backslash bei der Ersetzung vor der Interpretation durch die Shell. Ohne diesen Schutz geht die Bash sonst davon aus, dass dieses Kommando in der nächsten Zeile fortgesetzt wird.
- \a ist der *audible BEL* und gibt einen Piepton aus.
- \b ist der Platzhalter für *backspace*.
- \f bezeichnet das *form feed*-Zeichen.
- \n steht für *new line*.
- \r bezeichnet ein *return*-Zeichen.
- \t ist ein *horizontal tab*.
- \v bezeichnet den *vertical tab*.

Die folgende Befehlszeile entfernt mit der Bezeichnung `skript-laut` alle akustischen Signale aus dem Skript und erstellt aus dem Ergebnis das `skript-leise`:

```
archangel:/scriptdemo # cat skript-laut | tr -d \a >skript-leise
```

md5sum, sha256sum, sha512sum

Die drei in der Überschrift genannten Programme haben alle denselben Verwendungszweck und – was Sie freuen wird – dieselbe Syntax. Sie dienen dazu, Prüfsummen für Dateien zu berechnen oder Dateien anhand bestehender Prüfsummen zu überprüfen. Wenn Sie ISO-Dateien aus dem Internet herunterladen, sind Ihnen wahrscheinlich schon solche Prüfsummendateien begegnet. Der Grund hierfür ist, dass beim Download einer großen Datei diese möglicherweise beschädigt wird. Mithilfe der Prüfsummendatei können Sie nach dem Download die Integrität der Datei prüfen, so dass Sie keine bösen Überraschungen bei der Installation von einer beschädigten ISO-Datei erleben. Typische Namen dieser Dateien sind z. B. *MD5SUMS*, *SHA256SUMS* oder *SHA512SUMS*. Es kommen aber auch andere Dateinamen vor, die in der Regel mit den jeweiligen Prüfprogrammen korrespondieren. Für die LPIC-Prüfung müssen Sie aber nur diese drei kennen.

MD5 (*Message-Digest Algorithm 5*) ist eine ältere kryptographische Hashfunktion, die zunehmend von *SHA 2* (*Secure Hash Algorithm 2*) abgelöst wird. Die mathematische Funktionsweise dieser Algorithmen ist sehr komplex, aber Sie müssen sich damit auch nicht beschäftigen.

Dieses Beispiel zeigt die Erstellung einer Prüfsummendatei:

```
[root@arch-book]# sha512sum debian-9.3.0-amd64-DVD-1.iso > SHA512SUMS
```

Der Inhalt der Prüfsummendatei ist:


```
[harald@arch-book Downloads]$ cat SHA512SUMS
c20f347330b36bf5eeccdf6c9af45d20f7c81130fe04fd1704
ff54b78b544d2f40bedfdc4f4f9445ba3714ad0061f443fbb9b
fed99a0ce1c9235d4f05eb77d56
  debian-9.3.0-amd64-DVD-1.iso
```

Es werden ein Hashwert und ein Dateiname pro Zeile gespeichert. Das bedeutet, dass Sie für mehrere Dateien eine gemeinsame Prüfsummendatei erzeugen können. Es wir dann pro Datei eine Zeile generiert. Beispiel:

```
[root@arch-book Downloads]# sha512sum *.iso > SHA512SUMS
```

Um Dateien anhand der Prüfsummendatei zu prüfen, verwenden Sie die Option `-c` und geben Sie lediglich die Prüfsummendatei an:

```
[root@arch-book Downloads]# sha512sum -c SHA512SUMS
debian-9.3.0-amd64-DVD-1.iso: OK
opensuse-leap-42.3-dvd-x86_64.iso: OK
...
```

Wenn eine Datei Fehler aufweist, erhalten Sie eine entsprechende Fehlermeldung:

```
[harald@arch-book Downloads]$ sha512sum -c SHA512SUMS
debian-9.3.0-amd64-DVD-1.iso: FEHLSCHLAG
sha512sum: WARNUNG: die 1 berechnete Prüfsumme passte NICHT
```

103.3 Grundlegende Dateiverwaltung

Wichtung: 4

Beschreibung: Kandidaten sollten in der Lage sein, die grundlegenden Linux-Kommandos zur Verwaltung von Dateien und Verzeichnissen zu verwenden.

Wichtigste Wissensgebiete:

- einzelne Dateien und Verzeichnisse kopieren, verschieben und entfernen
- mehrere Dateien kopieren und Verzeichnisse rekursiv kopieren
- Dateien entfernen und Verzeichnisse rekursiv entfernen
- einfache und fortgeschrittene Dateinamen-Suchmuster in Kommandos verwenden
- `find` verwenden, um Dateien auf der Basis ihres Typs, ihrer Größe oder ihrer Zeitstempel zu finden und zu bearbeiten
- `tar`, `cpio` und `dd` verwenden

Liste wichtiger Dateien, Verzeichnisse und Anwendungen:

- `cp`
- `find`
- `mkdir`
- `mv`
- `ls`
- `rm`
- `rmdir`
- `touch`
- `tar`
- `cpio`
- `dd`
- `file`
- `gzip`
- `gunzip`
- `bzip2`
- `bunzip2`
- `xz`
- `unxz`
- *file globbing*

Allgemeines

In den folgenden Abschnitten geht es um die Grundlagen der Dateiverwaltung. Sie müssen für die Prüfung die grundlegenden Befehle des Dateimanagements und deren Optionen kennen. Außerdem müssen Sie in der Lage sein, Wildcards mit diesen Kommandos zu verwenden.

Kommandos für Dateioperationen

ls

Das Kommando `ls` listet den Inhalt von Verzeichnissen auf. Die Ausgabe erfolgt per Voreinstellung in alphabetischer Reihenfolge. In der Prüfung sollten Sie zumindest mit den folgenden Optionen vertraut sein:

- `-l` gibt an, dass das Listingformat verwendet wird. Dieses beinhaltet u. a. die Zugriffsberechtigungen auf Dateien und Verzeichnisse sowie Timestamps.
- `-i` zeigt die von Dateien und Verzeichnissen verwendeten Inodes an.

- `-a` listet alle Dateien auf, auch solche, die mit einem Punkt beginnen.
- `-s` zeigt in Kombination mit `-l` die Größe jeder Datei in Blocks an.
- `-h` zeigt die Dateigröße in einem menschenlesbaren Format an (21 KB, 24 MB, 3 GB usw.), setzt aber die Option `-l` voraus.

Wegen der Einfachheit des Programms verzichte ich hier auf umfangreiche Beispiele.

cd und pwd

Um das aktuelle Verzeichnis zu wechseln, verwenden Sie das Kommando `cd`. Bei `cd` handelt es sich um ein Shell-internes Kommando. Sie können sowohl absolute als auch relative Pfade mit diesem Befehl verwenden. Wenn Sie keinen Prompt eingestellt haben, der Ihnen das aktuelle Verzeichnis anzeigt, können Sie das Kommando `pwd` verwenden. Als Ergebnis wird Ihnen das aktuelle Verzeichnis angezeigt.

Die Tilde

Die Tilde (~) zeigt an, dass Sie sich gerade in Ihrem Heimatverzeichnis befinden. Sie können die Tilde auch in Pfadangaben verwenden. So würde das im folgenden Beispiel dargestellte Kommando die *dateiXY* in Ihrem Heimatverzeichnis löschen. Hierbei spielt es keine Rolle, in welchem Verzeichnis Sie sich gerade selbst befinden:

```
archangel:/ # rm ~/dateiXY
```

Wenn Sie in Ihr Heimatverzeichnis wechseln wollen, können Sie ebenfalls die Tilde verwenden. Die Eingabe von `cd` ohne Optionen und Argumente führt aber zu demselben Ergebnis:

```
harald@archangel:/> cd
harald@archangel:~>
```

Die Tilde in der zweiten Zeile zeigt, dass der Wechsel in das Heimatverzeichnis des Benutzers durchgeführt wurde.

Absolute Pfadangaben

Ein Verzeichniswechsel mit einer absoluten Pfadangabe beginnt immer mit einem Slash. Absoluter Pfad bedeutet, dass Sie den ganzen Pfad vom Hauptverzeichnis aus angeben, egal, in welchem Verzeichnis Sie sich selbst gerade befinden. Beispiel:

```
archangel:/textfiles # cd /var/log
archangel:/var/log #
```

Sie müssen also bei einem Verzeichniswechsel mit einer absoluten Pfadangabe Ihre eigene Position nie berücksichtigen.

Relative Pfadangaben

Bei einem Verzeichniswechsel mit relativer Pfadangabe bewegen Sie sich relativ zum aktuellen Verzeichnis fort. Eine relative Pfadangabe beginnt deshalb nie mit einem Slash. Beispiel:

```
archangel:/var # cd log
archangel:/var/log #
```

Es wurde hier relativ zum */var*-Verzeichnis in das */var/log*-Verzeichnis gewechselt.

Punkte

Zwei aufeinander folgende Punkte repräsentieren das übergeordnete Verzeichnis. Sie können also ohne weiteres durch die Eingabe von `cd ..` in das übergeordnete Verzeichnis wechseln. Beachten Sie, dass zwischen dem Befehl und den beiden Punkten ein Leerzeichen stehen muss. Das ist bei anderen Betriebssystemen teilweise anders. Ein einzelner Punkt repräsentiert das aktuelle Verzeichnis. Die Eingabe von `cd .` führt also zu keinem Ergebnis.

Prüfungstipp

Sie müssen in der Prüfung zwischen absoluten und relativen Pfadangaben unterscheiden können. Außerdem müssen Sie auf »exotische« Argumente für das `cd`-Kommando gefasst sein.

Beispiele für `cd`:

- `cd` wechselt in das Heimatverzeichnis.
- `cd /` wechselt in das Hauptverzeichnis.
- `cd ..` wechselt eine Verzeichnisebene höher.
- `cd ../..` wechselt zwei Verzeichnisebenen höher.
- `cd ../dateien` wechselt zuerst eine Verzeichnisebene höher und dann dazu relativ ins Verzeichnis *dateien*.

cp

Mit dem Kommando `cp` können Sie Dateien und Verzeichnisse kopieren. Sie müssen bei der Verwendung von `cp` immer sowohl eine Quelle als auch ein Ziel angeben. Das gilt auch, wenn Sie sich gerade im Zielverzeichnis für die Dateien befinden sollten. Letztes Argument muss hierbei immer das Ziel sein. Dafür können Sie aber mit einem einzigen Kommando mehrere Dateien kopieren (abgesehen von der Verwendung von Wildcards, siehe Abschnitt »Verwendung von Wildcards« auf Seite 124). Diese Mehrfachauswahl ist gleichzeitig der Grund, weshalb ein Ziel angegeben wer-

den muss. Die Shell könnte sonst bei der Interpretation des letzten Argumentes nicht wissen, ob es sich um eine Quelldatei oder um eine Zieldatei handelt. Folgende Punkte müssen bei der Verwendung von `cp` berücksichtigt werden:

- Sowohl die Quelle als auch das Ziel können wahlweise relativ oder absolut angegeben werden.
- Bei dem angegebenen Ziel kann es sich wahlweise um eine Datei oder auch um ein Verzeichnis handeln.
- Wurden mehrere Quelldateien angegeben, geht die Shell davon aus, dass es sich bei dem angegebenen Ziel um ein Verzeichnis handeln muss.
- Wenn es sich beim Kopieren einer einzigen Datei beim Ziel ebenfalls um eine Datei handelt und diese bereits existiert, wird die Zieldatei überschrieben. Sie können das automatische Überschreiben von Dateien mit der Option `-i` verhindern.
- Handelt es sich beim Kopieren einer einzigen Datei beim Ziel um ein Verzeichnis, dann wird die Datei in dieses Verzeichnis kopiert. Eventuell existierende Dateien gleichen Namens werden überschrieben.

Häufig verwendete Optionen von `cp` sind folgende:

- `-i` bzw. `--interactive` fragt vor dem Überschreiben eventuell existierender Dateien im Zielverzeichnis, ob diese Dateien wirklich überschrieben werden sollen.
- `-f` bzw. `--force` erzwingt den Schreibvorgang im Zielverzeichnis.
- `-p` bzw. `--preserve` kopiert die Datei(en) unter Beibehaltung des Eigentümers, der Eigentümergruppe, der Berechtigungen und der Timestamps.
- `-R`, `-r` bzw. `--recursive` kopiert ein Verzeichnis inklusive aller Unterverzeichnisse und Dateien.

In dem folgenden Beispiel werden die beiden Dateien */var/log/syslog* und */var/log/apache2/access.log* in das aktuelle Verzeichnis kopiert. Dieses wird durch den einzelnen Punkt am Ende der Befehlszeile repräsentiert:

```
archangel:~ # cp /var/log/syslog /var/log/apache2/access.log .
```

Beispiele mit alltäglichen Kopieraktionen möchte ich Ihnen an dieser Stelle ersparen. Sie sollten in der Prüfung darauf vorbereitet sein, Befehlszeilen wie die soeben dargestellte, inklusive der unmittelbar davor aufgeführten Optionen, lesen und verstehen zu können.

mv

Wenn Sie Dateien oder Verzeichnisse verschieben wollen, verwenden Sie das Kommando `mv`. Genauso wie `cp` benötigt `mv` eine Quellangabe und eine Zielangabe. Wenn

das Ziel sich auf derselben Partition befindet wie die Quelle, werden die Verzeichnisse und Dateien nicht tatsächlich verschoben. Es werden lediglich die Einträge in den jeweiligen Verzeichnissen geändert, um der Verschiebung Rechnung zu tragen. Wenn Dateien auf eine andere Partition verschoben werden, werden diese Dateien zunächst auf die andere Partition kopiert und die ursprünglichen Dateien hinterher gelöscht. Deshalb dauert das Verschieben von Daten über Partitionsgrenzen hinweg auch erheblich länger als Verschiebungen innerhalb einer Partition.

Häufig verwendete Optionen von `mv` sind folgende:

- `-u` bzw. `--update` erstellt eine Datei nur dann, wenn die bereits existierende Datei älter ist als die zu verschiebende Datei oder wenn die Zieldatei noch nicht existiert.
- `-i` bzw. `--interactive` fragt nach, ob bereits existierende Dateien wirklich überschrieben werden sollen. Per Voreinstellung wird nicht nachgefragt.
- `-f` bzw. `--force` erzwingt den Schreibvorgang im Zielverzeichnis.

mkdir

Um ein neues Verzeichnis zu erstellen, verwenden Sie das Kommando `mkdir`. Die zu erstellenden Verzeichnisse können relativ zum aktuellen Verzeichnis oder als absoluter Pfad angegeben werden. Häufig verwendete Optionen für `mkdir` sind folgende:

- `-p` bzw. `--parents` erstellt gegebenenfalls übergeordnete Verzeichnisse, wenn diese noch nicht existieren. Mit dieser Option kann eine ganze Verzeichnishierarchie mit nur einem Kommando erstellt werden.
- `-m` bzw. `--mode` stellt die Berechtigungen für das neue Verzeichnis während der Erstellung ein.

Mit dem folgenden Kommando erstellt der aktuell angemeldete Benutzer ein Verzeichnis, das nur er selbst verwenden kann. Voraussetzung für die Ausführung des Kommandos sind natürlich Schreibrechte im Zielverzeichnis.

```
archangel:/textfiles # mkdir -m 700 Gedichte
```

Die automatische Erstellung einer Verzeichnishierarchie könnte z. B. so aussehen wie in der folgenden Befehlszeile:

```
archangel:/ # mkdir --parents /arbeitsgruppe/dokumente/tabellen
```

rm

Mit dem Kommando `rm` können Sie Dateien und (mit den entsprechenden Optionen) Verzeichnisse löschen. Sie benötigen Schreibrechte in dem Verzeichnis, in dem Sie die Dateien löschen wollen, aber Sie benötigen keine Schreibrechte auf die Dateien selbst. Häufig verwendete Optionen von `rm` sind folgende:

- `-R`, `-r` bzw. `--recursive` löscht, auf ein Verzeichnis angewendet, auch dessen Unterverzeichnisse und alle Dateien.
- `-i` bzw. `--interactive` fragt bei jeder Datei noch einmal nach, ob sie wirklich gelöscht werden soll (ähnlich wie bei Windows).
- `-f` bzw. `--force` erzwingt das Löschen. Fehlermeldungen, die durch nicht vorhandene Dateien verursacht werden, werden unterdrückt.

Prüfungstipp

Rekursive Befehlsausführung ist ein beliebtes Prüfungsthema. Die Auswirkungen des Kommandos `rm -R` müssen Ihnen also klar sein.

rmdir

Das Kommando `rmdir` entfernt leere Verzeichnisse. Verzeichnisse, die noch Dateien enthalten, können auch mit zusätzlichen Optionen nicht gelöscht werden. Sie können die Option `-p` verwenden, damit übergeordnete Verzeichnisse automatisch mitgelöscht werden, wenn diese durch die Löschung des eigentlichen Verzeichnisses leer werden.

touch

Eigentlich wird das Kommando `touch` verwendet, um die Timestamps einer Datei zu ändern. Sie können `touch` aber auch verwenden, um neue Dateien mit einer Größe von 0 Byte zu erstellen. Gängige Optionen von `touch` sind:

- `-t` setzt den Timestamp auf den Wert [[CC]YY]MMDDhhmm[.ss].
- `-a` ändert nur die letzte Zugriffszeit.
- `-m` ändert nur die letzte Änderungszeit.

find

Das Programm `find` sucht nach Dateien in einer Verzeichnisstruktur. Um die Datei *textfile1* zu finden, geben Sie folgendes Kommando ein:

```
archangel:/ # find / -name textfile1
```

Der Slash direkt hinter dem eigentlichen Kommando weist `find` an, die Suche im Hauptverzeichnis zu beginnen. Die Option `-name` besagt, dass das Suchkriterium der Dateiname ist. Der Suchausdruck ist `textfile1`.

Die Thematik »Auffinden von Dateien« wird in einem späteren Kapitel noch sehr ausführlich beschrieben werden. Deshalb soll es hier zunächst bei dieser kurzen Erklärung bleiben.

Verwendung von Wildcards

Wildcards sind im Englischen die Joker eines Kartenspiels. Bei Linux werden Wildcards verwendet, um Operationen an mehreren Dateien gleichzeitig auszuführen, damit ein Kommando nicht für jede Datei einzeln eingegeben werden muss. Bei Suchoperationen dienen Wildcards auch dazu, Dateien zu spezifizieren, wenn deren Name nicht genau bekannt ist.

Die Wildcards werden durch die Shell interpretiert und nicht durch das jeweilige Kommando. Das bietet vor allem den Vorteil, dass die Verwendung der Wildcards bei allen unterstützten Programmen gleich abläuft.

Prüfungstipp

Beachten Sie, dass sich die Wildcards * und ? in der Interpretation von den entsprechenden DOS-Platzhaltern unterscheiden. Verwechseln Sie die Verwendungsmethoden nicht in der Prüfung.

Gängige Wildcards für Dateien sind:

- * ersetzt null oder mehr Zeichen. *Datei** beinhaltet demnach *Datei1*, *Datei255*, *Datei-A* usw. Im Gegensatz zur Ausgabe bei DOS-Wildcards ist auch *Datei* enthalten.
- ? ersetzt genau ein Zeichen. *Datei?* beinhaltet *Datei1*, *DateiX* usw., aber nicht *Datei*.
- `[a-z]` ersetzt ein einzelnes Zeichen aus dem angegebenen Bereich. Die Suche nach *Datei[a-d]* beinhaltet also genau *Dateia*, *Dateib*, *Dateic* und *Dateid*. Dasselbe Verfahren ist auch mit Großbuchstaben oder Ziffern durchführbar.
- `[abcde]` ersetzt ein einzelnes der angegebenen Zeichen. Den Unterschied zu `[a-z]` verdeutlicht dieses Beispiel: *Datei[amz]* liefert die Dateien *Dateia*, *Dateim* und *Dateiz*.
- `[!a-z]` ersetzt ein einzelnes Zeichen, das nicht angegeben ist.

Beispiele für die Verwendung von Wildcards sind folgende:

```
archangel:~ # find / -name "*.log"
```

Damit erfolgt das Durchsuchen des Systems nach Logdateien.

```
archangel:~ # find / -name "samba*.rpm"
```

Diese Wildcard bewirkt die Suche nach *RPM*-Paketen, die mit Samba in Zusammenhang stehen.

Komprimierung und Archivierung

tar (Tape Archiver)

Das Archivierungsprogramm `tar` wurde eigentlich entwickelt, um Dateien zu einem Archiv zusammenzufassen und dieses Archiv dann auf ein Bandlaufwerk zu sichern. Sie sollten sich merken, dass man bei der Übergabe von Optionen an `tar` einen Strich verwenden kann, aber nicht muss. In Bezug auf das erste Beispiel bedeutet das, dass die folgenden Kommandos beide zulässig sind und dasselbe Ergebnis zur Folge haben:

```
archangel:/usr/src # tar -xvzf xmbmon205.tar.gz
archangel:/usr/src # tar xvzf xmbmon205.tar.gz
```

Sie können also bei sogenannten »Fill in the Blanks«-Prüfungsaufgaben beide Schreibweisen verwenden. Wichtige Optionen sind folgende:

- `-x` (extract) extrahiert ein Archiv.
- `-z` (gzip/gunzip) sorgt für eine Komprimierung mittels `gzip` bzw. eine Entkomprimierung durch `gunzip`.
- `-v` (verbose) schaltet den Verbose-Mode ein.
- `-c` (create) erstellt ein neues Archiv.
- `-t` (table) listet den Inhalt eines Archivs auf.
- `-j` (bzip2/bunzip2) sorgt für eine Komprimierung mittels `bzip2` bzw. eine Entkomprimierung durch `bunzip2`.
- `-f` (file) zeigt an, dass zur Eingabe bzw. Ausgabe eine Datei verwendet wird.

Sie sollten für die Prüfung wenigstens die oben genannten Optionen kennen. Die Reihenfolge der Optionen spielt, wie bei den meisten Programmen, keine Rolle. Eine Ausnahme ist die Option `f`. Diese muss immer als letzte verwendet werden.

Das folgende Beispiel erzeugt ein mit `bzip2` komprimiertes Archiv der Konfigurationsdateien eines Linux-Systems:

```
archangel:/ # tar -cvjf backup.tar.bz2 /etc/*
```

Um den Inhalt eines solchen Archivs zu betrachten, benötigen Sie folgendes Kommando:

```
archangel:/ # tar -tvjf backup.tar.bz2 | less
```

Im Notfall können die Dateien ohne Probleme wiederhergestellt werden:

```
archangel:/ # tar -xvjf backup.tar.bz2
```

gzip

Wenn Sie Dateien komprimieren wollen, können Sie das Programm `gzip` verwenden. Wie Sie im vorangegangenen Abschnitt gelesen haben, kann `tar gzip` aufrufen. Wenn Sie mehrere Dateien zu einem Archiv zusammenfassen wollen, ist `tar` in Kombination mit den Optionen `z` oder `j` die bessere Lösung. Wenn `gzip` ohne Optionen verwendet wird, komprimiert es die angegebenen Dateien, hängt an die Zieldateien die Erweiterung *.gz* an und löscht anschließend die Originaldateien. Sollen die Originaldateien erhalten bleiben, müssen Sie die Option `-c` verwenden. Damit man die Effizienz von `gzip` sehen kann, wird diese Option im folgenden Beispiel angewendet:

```
archangel:/demo # gzip lpi101.doc -c > lpi101.doc.gz
archangel:/demo # ls -lh
total 1.2M
drwxr-xr-x   2 root root 4.0K Mär 18 17:50 .
drwxr-xr-x  24 root root 4.0K Mär 18 17:45 ..
-rwxr--r--   1 root root 903K Mär 18 17:46 lpi101.doc
-rw-r--r--   1 root root 237K Mär 18 17:50 lpi101.doc.gz
```

Die Datei wurde von 903 KB auf 237 KB komprimiert. Der Redirektor > war in diesem Fall notwendig, weil die Ausgabe von `gzip` auf dem Bildschirm erfolgt, wenn die Option `-c` verwendet wird. Um eine Datei zu komprimieren, ohne das Original beizubehalten, geben Sie die Datei einfach ohne weitere Parameter an:

```
archangel:/demo # gzip lpi101.doc
```

Sie können auch mehrere Dateien durch Leerstellen voneinander getrennt angeben oder Platzhalter (Wildcards) verwenden.

Den Inhalt eines mit `gzip` komprimierten Archivs können Sie mit dem Schalter `-l` auflisten. Es wird dann auch die komprimierte Größe der Einzeldateien angezeigt, falls das Archiv mehrere Dateien enthält:

```
archangel:/demo # gzip -l lpi101.doc.gz
         compressed        uncompressed  ratio uncompressed_name
                 31                   0   0.0 % lpi101.doc
```

gunzip

Um eine oder mehrere Dateien, die mit `gzip` komprimiert wurden, wieder zu entkomprimieren, stehen Ihnen mehrere Möglichkeiten zur Verfügung:

- `gunzip` ist das offizielle Gegenstück zu `gzip`. Auf der Kommandozeile angegebene Dateien werden von `gunzip` dekomprimiert.
- `gzip -d` startet `gzip` im Modus *decompress*.
- `zcat` verhält sich ohne Optionen wie `gunzip -c`.

bzip2

Sie können zum Komprimieren von Dateien auch `bzip2` verwenden. In vielen Fällen erreichen Sie hierdurch eine höhere Kompression als mit `gzip`.

```
archangel:/demo # bzip2 lpi101.doc -c > lpi101.doc.bz2
archangel:/demo # ls -lh
total 1.2M
drwxr-xr-x   2 root root 4.0K Mär 18 18:00 .
drwxr-xr-x  24 root root 4.0K Mär 18 17:45 ..
-rwxr--r--   1 root root 903K Mär 18 17:46 lpi101.doc
-rw-r--r--   1 root root 213K Mär 18 18:00 lpi101.doc.bz2
```

Die Datei wurde von 903 KB auf 213 KB komprimiert. Die Effizienz war also zumindest in diesem Fall etwas besser als bei der Verwendung von `gzip`.

bunzip2

Genauso wie bei `gzip` gibt es auch zur Entkomprimierung von *bz2*-Dateien folgende Möglichkeiten:

- `bunzip2`
- `bzip2 -d`
- `bzcat`

Tatsächlich handelt es sich in allen drei Fällen um dasselbe Programm, das lediglich jeweils über unterschiedliche Links aufgerufen wird.

xz

Mit diesem Programm erstellen Sie komprimierte Archive mit der Dateierweiterung `xz`. Die Verwendung ist ähnlich wie bei den Programmen `gzip` und `bzip2`. Dementsprechend heißen die Gegenstücke zu `xz` jeweils `unxz` und `xzcat`. Sie können u. a. folgende gängige Optionen mit `xz` verwenden:

```
-z, --compress
-d, --decompress, --uncompress
-t, --test
-l, --list
```

Die Syntax der Optionen ist jeweils selbsterklärend.

Verwendung von cpio

Das Programm `cpio` wird verwendet, um Dateien in Archive hinein- oder aus Archiven herauszukopieren. Es ähnelt vom Verwendungszweck also im weitesten Sinne dem `tar`-Befehl. Eigentlich wird `cpio` heutzutage kaum noch verwendet, aber Sie

müssen die grundlegenden Optionen für die Prüfung kennen. Es gibt drei verschiedene Betriebsmodi, in denen `cpio` arbeitet. Welcher Modus verwendet wird, hängt von der ersten übergebenen Option ab. Die Benennung dieser drei Schalter scheint auf den ersten Blick recht paradox. Merken Sie sich bitte trotzdem mindestens diese Optionen:

- `cpio -i` bzw. `--extract` aktiviert den Copy-in-Modus. In diesem Modus werden Dateien aus einem Archiv in das Dateisystem kopiert. Sie müssen also das `-i` aus Sicht des Dateisystems sehen und nicht aus Sicht des Archivs.
- `cpio -o` bzw. `--create` steht für das Gegenteil von `-i` und wird als Copy-out-Modus bezeichnet. Es werden also diesmal Dateien aus (out) dem Dateisystem in das Archiv kopiert bzw. zunächst ein Archiv erstellt (create).
- `cpio -p` bzw. `--pass-through` – in diesem Modus wird weder ein Archiv erstellt noch überhaupt ein Archiv verwendet. Das Programm `cpio` liest die zu kopierenden Dateien vom Standardeingabekanal und kopiert diese dann in ein Verzeichnis, das als Argument übergeben wurde.

Ein Beispiel, das ähnlich auch in der Prüfung vorkommen könnte, verlangt, dass alle mit OpenOffice.org erstellten Tabellendokumente, die in den Heimatverzeichnissen der Benutzer existieren, in ein Verzeichnis */backup* gesichert werden sollen. Die im Dateisystem verwendete Verzeichnisstruktur soll im Backup nachvollziehbar sein. Der verwendete Befehl könnte dann so aussehen:

```
archangel:/ # find /home -name *.ods | cpio -pd /backup
```

Das Suchwerkzeug `find` sucht hier also unterhalb von */home* nach Dateien mit der Erweiterung *ods* und übergibt diese dann an das Programm `cpio`. Die Option `-p` bringt `cpio` in den Passthrough-Modus. Es wird also kein Archiv angelegt. Das Kommando `-d` sorgt dafür, dass die benötigten Directories angelegt werden. Als letztes Argument gibt */backup* das Ziel für `cpio` an.

Die von `cpio` verwendeten Optionen würden ohne Probleme zwei Buchseiten füllen. Deshalb verweise ich hier auf die Manpages zu `cpio`. Für die Prüfung reicht es normalerweise aus, die Schalter für die Betriebsmodi von `cpio` zu kennen.

Verwendung von dd

Mit `dd` können Sie Daten kopieren oder konvertieren. Da die Konvertierung von Daten in diesem Zusammenhang keine Rolle spielt, sei nur erwähnt, dass es etwa möglich ist, Daten von einem Datenträger zu lesen und dann mit einer geänderten Blockgröße auf einen anderen Datenträger zu schreiben.

Wenn Sie `dd` (eigentlich sinnloserweise) ohne Parameter starten, dann liest dieser von der Standardeingabe und schreibt auf die Standardausgabe. Sie können dann

erkennen, dass `dd` »stur« das schreibt, was Sie eingegeben haben, ohne zu interpretieren oder Modifikationen vorzunehmen:

```
archangel:/ # dd
Mal was schreiben und mit Strg + D abschicken...
Mal was schreiben und mit Strg + D abschicken...
0+1 records in
0+1 records out
49 bytes (49 B) copied, 4.2909 seconds, 0.0 kB/s
```

Diese Eigenschaft ist auch schon einer seiner großen Vorzüge, weil Sie auf diese Art problemlos empfindsame Betriebssysteme von einer Festplatte zu einer anderen klonen können. Es ist ratsam, gleich große Festplatten zu verwenden, weil selbst der *MBR* und die darin enthaltene Partitionstabelle mit kopiert werden. Dazu müssen natürlich der Eingabekanal und der Ausgabekanal geändert werden. Wenn die Originalfestplatte */dev/sda* ist und die erwünschte Kopie an */dev/sdb* angeschlossen wurde, sieht die `dd`-Zeile so aus:

```
archangel:/ # dd if=/dev/sda of=/dev/sdb
```

In diesem Zusammenhang möchte ich darauf hinweisen, dass die Duplizierung großer Datenträger mit `dd` erhebliche Zeit in Anspruch nimmt.

Sehr beliebt ist auch das Sichern des Master Boot Records mittels `dd`. Das funktioniert dann so:

```
archangel:/ # dd if=/dev/sda of=mbr.backup ibs=512 count=1
1+0 records in
1+0 records out
512 bytes (512 B) copied, 0.037257 seconds, 13.7 kB/s
```

Hier wurde als Eingabe die Festplatte an */dev/sda* verwendet. Die Ausgabe erzeugt die Datei *mbr.backup* im aktuellen Verzeichnis, weil kein Pfad angegeben wurde. Der Parameter `ibs` steht für *in block size* und sorgt dafür, dass ein gelesener Block immer 512 Byte groß ist. Mit `count` wird die Anzahl der zu lesenden Blöcke auf `1` festgelegt, denn der *MBR* ist ja ausschließlich der erste Block auf einer Festplatte.

Soll `dd` zur Erstellung einer Datensicherung verwendet werden, müssen Sie als Ausgabekanal einfach den Streamer einstellen. Denken Sie hierbei immer daran, dass es rückspulende und nicht rückspulende Geräte gibt. Beispiel:

```
archangel:/ # dd if=/home of=/dev/st0 cbs=16b
```

Dieses Kommando sichert alle Benutzerverzeichnisse auf den ersten SCSI-Streamer und spult das Band anschließend zurück. Der Wert `cbs` sorgt für eine dem Streamer angemessene Blockgröße.

file

Mit dem Kommando `file` können Sie bei einer unbekannten Datei feststellen, um welchen Dateityp es sich handelt. Am besten sehen Sie den Verwendungszweck an ein paar Beispielen:

```
root@archangel:~# file /bin/bash
/bin/bash: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32,
BuildID[sha1]=1038e837cd34f24262050f0f4afb8a1fa66b32a3, stripped
```

Über */bin/bash* wissen Sie ja bereits bestens Bescheid. Deshalb ist dieses Beispiel auch selbsterklärend. In den nächsten Zeilen stößt `file` auf ein gewöhnliches Shell-Skript:

```
root@archangel:~# file /etc/cron.daily/backup
/etc/cron.daily/backup: Bourne-Again shell script text executable
```

[+]

Prüfungstipp

Sie können `file` sehr viele Optionen übergeben (siehe Manpage). Für die Prüfung reicht es aber völlig aus zu wissen, was das Programm grundsätzlich macht.

103.4 Ströme, Pipes und Umleitungen verwenden

Wichtung: 4

Beschreibung: Kandidaten sollten in der Lage sein, Ströme umzuleiten und zu verbinden, um Textdaten effizient zu verarbeiten. Zu diesen Aufgaben gehören das Umleiten der Standardeingabe, Standardausgabe und Standardfehlerausgabe, das Weiterleiten der Ausgabe eines Kommandos an die Eingabe eines anderen Kommandos, die Verwendung der Ausgabe eines Kommandos als Argumente für ein anderes Kommando und das Senden der Ausgabe sowohl an die Standardausgabe als auch an eine Datei.

Wichtigste Wissensgebiete:

- Umleiten der Standardeingabe, Standardausgabe und Standardfehlerausgabe
- Weiterleiten der Ausgabe eines Kommandos an die Eingabe eines anderen Kommandos (Pipe)
- Verwenden der Ausgabe eines Kommandos als Argumente für ein anderes Kommando
- Senden der Ausgabe sowohl an die Standardausgabe als auch an eine Datei

Liste wichtiger Dateien, Verzeichnisse und Anwendungen:

- `tee`
- `xargs`

Allgemeines

Manchmal kann es vorkommen, dass Sie die Ausgabe eines Programms direkt mit einem anderen Programm weiterverarbeiten müssen. In diesem Fall können Sie eine sogenannte *Pipe* (senkrechter Strich) verwenden. Mit ihrer Hilfe ist das Zwischenspeichern der Ausgabe des ersten Programms in einer Datei zum Zweck, diese Datei dann mit dem zweiten Programm wieder einzulesen, unnötig.

In einem anderen Fall erzeugt ein Programm vielleicht eine Ausgabe auf dem Bildschirm, die Sie abspeichern wollen. In diesem Fall benötigen Sie eine Umleitung der Standardausgabe (Redirect).

Wenn ein Programm ausschließlich von der Standardeingabe (normalerweise die Tastatur) liest und Sie mit diesem Programm eine Datei einlesen müssen, benötigen Sie ebenfalls eine Umleitung. Diesmal muss allerdings die Standardeingabe umgeleitet (Redirect) werden.

stdin, stdout und stderr

Immer wenn unter Linux ein Programm ausgeführt wird, erhält dieses Informationen über drei Dateideskriptoren. Diese werden als Standard-I/Os bezeichnet:

- *Standardeingabekanal* (*stdin*) ist normalerweise die Tastatur. Viele Programme erwarten ihre Eingaben von *stdin*. Es gibt aber auch, wie Sie schon oft gesehen haben, Programme, die stattdessen Dateien als Argumente erwarten. Diese Programme verwenden *stdin* nicht.

  stdin entspricht dem Dateideskriptor 0.
- *Standardausgabekanal* (*stdout*) ist normalerweise ein Terminal. Viele Programme machen ihre Ausgaben direkt nach *stdout*.

  stdout entspricht dem Dateideskriptor 1.
- *Standardfehlerkanal* (*stderr*) ähnelt vom Verhalten her *stdout*, enthält aber nur die Fehlermeldungen eines Programms. Die Ausgabe des Fehlerkanals erfolgt normalerweise auch auf dem Terminal.

  stderr entspricht dem Dateideskriptor 2.

Dadurch, dass der Standardausgabekanal und der Standardfehlerkanal getrennt verwaltet werden, ist es möglich, Fehlermeldungen von den normalen Ausgaben eines Programms zu trennen. Deshalb können Sie den Fehlerkanal z. B. in eine Fehlerprotokolldatei umlenken.

Umleitungen (Redirects)

Umleitungen werden verwendet, um die Standard-I/Os entweder in eine Datei hinein oder aus einer Datei heraus umzulenken. Das ist z. B. dann erforderlich, wenn man einem Programm keine Dateien als Argumente übergeben kann. Wenn Sie alle Meldungen des Kernels in einer separaten Datei speichern möchten, können Sie mittels `grep` im *Syslog* nach solchen Meldungen suchen und diese dann mit einem Redirektor in eine andere Datei speichern:

```
archangel:/diag # grep kernel /var/log/syslog > kernelmessages
```

Mit einem solchen Kommando erfassen Sie nur die Standardausgabe von `grep`. Tritt ein Fehler auf, werden die entsprechenden Meldungen weiterhin auf der Konsole ausgegeben. Sie können mit einer Kommandozeile wie der folgenden Standardmeldungen und Fehlermeldungen in zwei unterschiedlichen Dateien aufzeichnen:

```
archangel:/scripts # script-xy 2>fehler.log 1>erfolg.log
```

Hierbei lenken das Argument `1>erfolg.log` die normalen Meldungen des Programms und `2>fehler.log` die Fehlermeldungen in je eine Datei um. Wenn nur eine einzige Datei für die Aufzeichnung sowohl von *stdout* als auch *stderr* verwendet werden soll, können Sie auch ein Kommando wie das folgende verwenden:

```
archangel:/scripts # script-xy >protokolldatei 2>&1
```

Der erste Teil des Kommandos `script-xy >protokolldatei` sorgt für die Umleitung von *stdout* in die Textdatei. `2>&1` leitet *stderr* auf *stdout* um. Das `&` besagt, dass `1` keine Datei ist. Das Kommando würde ohne das `&` den Standardfehlerkanal in die Datei *1* umleiten.

Wenn Sie einen Standard-I/O umleiten, wird die Zieldatei, wenn diese noch nicht existiert, automatisch erstellt. Sollten Sie dieselbe Umleitung noch einmal durchführen, wird die ursprüngliche Datei überschrieben. Um Daten an eine bestehende Datei anzuhängen, die für Umleitungen verwendet wird, müssen Sie einfach zwei Redirektorzeichen verwenden (`>>`).

```
archangel:/diag # grep kernel /var/log/syslog >> kernelmessages
```

Wenn Sie nur einen einzelnen Redirektor verwenden, wird gleich zu Beginn der Programmausführung die Zieldatei erstellt. Ist diese bereits vorhanden, wird sie gelöscht und neu erstellt. Sollten also Zieldatei und Eingabedatei identisch sein, ist ein Datenverlust garantiert. Das folgende Kommando hat demnach eine leere Kundendatenbank zur Folge:

```
archangel:/db # grep "Meier" kunden-db > kunden-db
```


Es besteht aber auch die Möglichkeit, den Standardeingabekanal umzuleiten. Beliebt hierfür ist das Programm `mail`. Das folgende Kommando sendet den kompletten *Syslog* per Mail an einen kompetenten Mitarbeiter. Der Betreff lautet »Überprüfen!« und der Inhalt der Datei erscheint im Textkörper der Mail:

```
archangel:/ # mail -s "Überprüfen!" willi < /var/log/syslog
```

Zusammenfassung:

- Um die Standardausgabe umzulenken, verwenden Sie diese Syntax:
  `Kommando > Zieldatei` (überschreibend)
  `Kommando 1> Zieldatei` (überschreibend)
  `Kommando >> Zieldatei` (anhängend)
  `Kommando 1>> Zieldatei` (anhängend)
- Um den Standardfehlerkanal umzulenken, verwenden Sie:
  `Kommando 2> Zieldatei` (überschreibend)
  `Kommando 2>> Zieldatei` (anhängend)
- Um beide Ausgabekanäle umzulenken, verwenden Sie:
  `Kommando > Zieldatei 2>&1` (gemeinsame Zieldatei)
  `Kommando 1> ZieldateiA 2> ZieldateiB` (getrennte Zieldateien)
- Um die Standardeingabe umzulenken, verwenden Sie diese Syntax:
  `Kommando < Quelldatei`

Pipes

Im Gegensatz zu Umleitungen lenken Pipes Datenströme nicht in Dateien um oder aus Dateien heraus. Sie sorgen vielmehr dafür, dass die Ausgabe eines Programms direkt als Eingabe für ein anderes Programm verwendet werden kann. Die Ausgabe des ersten Programms erfolgt dann nicht nach *stdout*. Oft werden Umleitungen und Pipes auch miteinander kombiniert. Sehr gebräuchlich ist die Umlenkung größerer Textmengen nach `less`. Das folgende Kommando würde möglicherweise eine solche größere Textmenge verursachen:

```
archangel:/ # grep "kernel" /var/log/syslog
```

Damit Sie die Ausgabe des Kommandos überhaupt komplett lesen können, wird sie einfach mit einer Pipe an `less` weitergegeben:

```
archangel:/ # grep "kernel" /var/log/syslog | less
```

Eine Kombination aus Umleitungen und Pipes verwendet z. B. üblicherweise das Programm `tr`. Das liegt daran, dass `tr` weder eine Eingabedatei noch eine Ausgabedatei

als Argument übergeben werden kann. Deshalb erfolgt die Eingabe oft mittels einer Pipe aus dem Programm `cat` heraus und die Ausgabe mit einer Umlenkung in die entsprechende Zieldatei:

```
archangel:/textfiles # cat textfile1 | tr \n \r > textfile2
```

tee und xargs

Das Programm `tee` hat nichts mit dem gleichnamigen aromatischen Aufgussgetränk zu tun, sondern eher mit einem T-Stück. Mit `tee` kann der Datenstrom eines Programms gleichzeitig auf der Konsole und in einer Textdatei ausgegeben werden. Beispiel:

```
archangel:/ # grep pppd /var/log/syslog | tee pppdmessages
```

Mit `xargs` können Sie die Ergebnisse eines Programms, das eine mehrzeilige Ausgabe liefert, an ein Programm übergeben, das immer nur ein Argument gleichzeitig verarbeiten kann. Beispiel:

```
archangel:/ # cut -d " " -f1 /var/log/apache2/access.log |\
sort | uniq | xargs -n1 host
```

Das Kommando wurde der Übersichtlichkeit halber mit dem Backslash auf zwei Zeilen aufgeteilt. Zur Erklärung: Mit diesem Kommando soll überprüft werden, von welchen Domänen aus auf einen Webserver zugegriffen wurde. Zunächst wird mit dem `cut`-Befehl die erste Spalte des *access.log* von Apache isoliert. Diese Spalte enthält die IP-Adressen der Besucher. Das Kommando `sort` sortiert die Liste der IP-Adressen. Mit `uniq` wird dafür gesorgt, dass jede IP-Adresse nur einmal in der Liste auftaucht. Diese Liste kann nicht direkt an das Kommando `host` zur Namensauflösung übergeben werden, weil `host` nur ein einzelnes Argument erwartet. Deshalb übergibt `xargs` die IP-Adressen einzeln. Das Kommando `host` wird also durch `xargs` für jede IP-Adresse einmal aufgerufen.

103.5 Prozesse erzeugen, überwachen und beenden

Wichtung: 4

Beschreibung: Kandidaten sollten die einfache Prozessverwaltung beherrschen.

Wichtigste Wissensgebiete:

- Jobs im Vordergrund und im Hintergrund ablaufen lassen
- einem Programm signalisieren, dass es nach dem Abmelden weiterlaufen soll

- aktive Prozesse beobachten
- Prozesse zur Ausgabe auswählen und sortieren
- Signale an Prozesse schicken

Liste wichtiger Dateien, Verzeichnisse und Anwendungen:

- `&`
- `bg`
- `fg`
- `jobs`
- `kill`
- `nohup`
- `ps`
- `top`
- `free`
- `uptime`
- `pgrep`
- `pkill`
- `killall`
- `watch`
- `screen`
- `tmux`

Allgemeines

In diesem und auch im nächsten Kapitel geht es um die Steuerung und Überwachung von Prozessen. Jedes Programm, das auf einem Computer läuft, besteht aus mindestens einem Prozess. Prozesse sind in einer Baumhierarchie angeordnet und die Wurzel dieses Baumes ist entweder der Prozess `init` oder `systemd`. Wenn der Kernel beim Systemstart alle seine Module initialisiert hat, übergibt er die Kontrolle an den jeweiligen init-Prozess. Jedem Prozess wird eine eindeutige ID zugeordnet, die *PID* (Prozess-ID). Der *init*-Prozess hat immer die *PID* 1. Die *PID* 0 gibt es nicht. Fallen Sie in der Prüfung nicht darauf herein!

```
root@ubuntu-server:~# ps -A | head -2
  PID TTY          TIME CMD
    1 ?        00:00:04 init
```

Alternativ auf einem Computer mit *systemd*:

```
root@archangel:~# ps -A|head -2
  PID TTY          TIME CMD
    1 ?        00:00:02 systemd
```

Der *init*-Prozess ist der erste Prozess, der auf einem System langfristig gestartet wird, und der letzte Prozess, der beim Herunterfahren des Systems stirbt.

Überwachen von Prozessen

ps

Mit dem Kommando `ps` können Sie die aktuell auf einem Computer laufenden Prozesse anzeigen. Es handelt sich hierbei allerdings lediglich um eine Momentaufnahme. Programme, die über eine kurze Lebensdauer verfügen (z. B. `df`), werden Sie mit `ps` nicht anzeigen können. Wenn Sie das Kommando `ps` ohne Optionen ausführen, erhalten Sie lediglich eine Liste der Prozesse, die auf dem aktuellen Terminal laufen. Das sind in der Regel nur die Shell und das `ps`-Kommando selbst. Wenn Jobs im Hintergrund laufen (oder stehen), werden diese auch angezeigt. Sie können bei der Übergabe von Optionen einen Strich voranstellen oder auch nicht. In Abhängigkeit von der verwendeten Programmversion werden Sie zu unterschiedlichen Ergebnissen kommen. Nach der UNIX-Syntax wird der Option ein Strich vorangestellt, während nach BSD-Syntax kein Strich verwendet wird. GNU-Optionen sind wiederum die ausgeschriebenen langen Optionen. Diesen werden zwei Striche vorangestellt. Am besten lesen Sie im Zweifelsfall immer in der Manpage der gerade verwendeten Version von `ps` nach.

Optionen können, wie bei den meisten Programmen, nach Belieben kombiniert werden. Folgende Optionen sind sowohl für die Prüfung als auch für die Praxis wichtig:

- `-a` zeigt auch die Prozesse anderer Benutzer an. Voraussetzung ist, dass diese Prozesse mit einem Terminal verknüpft sind.
- `-u` zeigt auch die Startzeit, den Pfad zur ausführbaren Datei und den ausführenden Benutzer an.
- `-x` führt auch die Prozesse auf, die nicht mit einem Terminal verbunden sind (z. B. *init* oder *cron*).
- `-C prozess` sorgt für die Ausgabe aller Instanzen eines auf der Kommandozeile angegebenen Prozesses.
- `-U benutzer` zeigt die Prozesse eines bestimmten Benutzers an.

Es folgen einige typische Beispiele, die ähnlich auch in der Prüfung Verwendung finden können.

Um alle Instanzen des *nfsd* (*NFS-Daemon*), die auf einem System laufen, anzuzeigen, geben Sie folgendes Kommando ein:


```
root@ubuntu-server:~# ps -C nfsd
  PID TTY          TIME CMD
 9592 ?        00:00:00 nfsd
 9593 ?        00:00:00 nfsd
 9594 ?        00:00:00 nfsd
... weitere Zeilen wurden abgeschnitten ...
```

Eine sehr beliebte Kombination ist `-aux`. Damit werden alle Prozesse sämtlicher Benutzer angezeigt. Zusätzlich werden Prozesse gelistet, die nicht mit einem Terminal verbunden sind. Die Ausgabe enthält umfangreiche Informationen zum Prozess:

```
root@ubuntu-server:~# ps aux
USER  PID %CPU %MEM    VSZ   RSS TTY  STAT START TIME COMMAND
root    1  0.0  0.3   2912  1848 ?  Ss Oct16 0:04 /sbin/init
root    2  0.0  0.0      0     0 ?  S  Oct16 0:00 [migration/0]
root    3  0.0  0.0      0     0 ?  SN Oct16 0:00 [ksoftirqd/0]
root    4  0.0  0.0      0     0 ?  S  Oct16 0:00 [watchdog/0]
... weitere Zeilen wurden abgeschnitten ...
```

Wenn Sie in einer langen Prozessliste schnell nach einem laufenden Prozess suchen müssen, empfiehlt es sich, die Ausgabe des Kommandos an `grep` weiterzuleiten:

```
root@ubuntu-server:~# ps -A | grep apache
  398 ?        00:00:00 apache2
  404 ?        00:00:00 apache2
... weitere Zeilen wurden abgeschnitten ...
```

pstree

Das Kommando `pstree` zeigt ebenfalls die aktuell auf einem Computer laufenden Prozesse an. Im Gegensatz zu `ps` zeigt `pstree` aber auch die Hierarchie der Prozesse in einem Baumdiagramm aus ASCII-Zeichen an. Die Ausgabe des Kommandos sieht folgendermaßen aus:

```
root@archangel:~# pstree
systemd─┬─ModemManager─┬─{gdbus}
        |              └─{gmain}
        ├─NetworkManager─┬─{NetworkManager}
        |                ├─{gdbus}
        |                └─{gmain}
        ├─accounts-daemon─┬─{gdbus}
        |                 └─{gmain}
        ├─agetty
        ├─apache2───3*[apache2]
        ├─at-spi-bus-laun─┬─dbus-daemon
```

```
        |                 ├─{dconf worker}
        |                 ├─{gdbus}
        |                 └─{gmain}
        ├─at-spi2-registr──{gdbus}
        ├─atd
... weitere Zeilen wurden abgeschnitten ...
```

Folgende Optionen, die zum Teil schon im vorangegangenen Beispiel enthalten sind, sollten Sie kennen:

- `-a` zeigt zusätzlich die Optionen und Argumente an, die einem Prozess an der Kommandozeile übergeben wurden.
- `-G` hat eine Ausgabe im VT100-Modus zur Folge. Das führt bei den meisten Terminals zu einer optisch ansprechenderen Ausgabe.
- `-p` zeigt zusätzlich die *PIDs* an.
- `-n` sorgt für eine Sortierung nach *PIDs*. Normalerweise gibt `pstree` die Prozesse in alphabetischer Reihenfolge aus.

top

Mit `top` können Sie die auf einem Computer laufenden Prozesse in Echtzeit überwachen. Während der Ausführung werden die folgenden Informationen dynamisch im Terminal ausgegeben:

- allgemeine Informationen, wie Uhrzeit, Uptime, Anzahl der angemeldeten Benutzer, durchschnittliche Systemauslastung
- Tasks: Anzahl der Prozesse insgesamt, laufende Prozesse, schlafende Prozesse, gestoppte Prozesse und Zombie-Prozesse
- CPU(s): Benutzung durch Benutzerprozesse, Systemprozesse, den Idle-Prozess u. a.
- Mem: Speicher insgesamt, in Verwendung, freier Speicher und für Buffers verwendeter Speicher
- Swap: diverse Werte zur Verwendung der Swap-Partition(en)

Anschließend folgt eine Auflistung der laufenden Prozesse. Diese werden standardmäßig in Reihenfolge der CPU-Verwendung ausgegeben. Die inaktiven Prozesse werden anschließend in der Reihenfolge der *PIDs* ausgegeben. Beachten Sie bitte, dass das Programm `top` selbst auch Systemressourcen belegt und deshalb das Messergebnis beeinflusst. Dies demonstriert das folgende Beispiel:

```
PID USER   PR  NI  VIRT  RES  SHR S %CPU %MEM   TIME+  COMMAND
223 root   15   0 2056 1016 752 R  1.5  0.0   0:00.03 top
 1 root    16   0  680  104  72 S  0.0  0.0   0:03.71 init
```



```
 2 root    RT  0    0    0    0 S  0.0  0.0   0:04.70 migration/0
 3 root    34 19    0    0    0 S  0.0  0.0  11:55.58 ksoftirqd/0
 4 root    RT  0    0    0    0 S  0.0  0.0   0:16.45 migration/1
 5 root    34 19    0    0    0 S  0.0  0.0   3:09.48 ksoftirqd/1
 6 root    10 -5    0    0    0 S  0.0  0.0   0:04.55 events/0
```

Sie können `top` sowohl beim Aufruf als auch zur Laufzeit Optionen übergeben. Für die Prüfung müssen Ihnen die gängigsten Optionen für beide Situationen bekannt sein.

Wichtige Interaktivoptionen:

- `k` (kill) tötet einen Prozess. Es müssen sowohl die *PID* als auch das Signal angegeben werden.
- `n` (number of) nennt die Anzahl der Zeilen, die `top` ausgibt.
- `r` (renice) ändert den Nice-Wert eines Prozesses zur Laufzeit.
- `h` (help) gibt eine Hilfe aus.
- `q` (quit) beendet das Programm.

Wichtige Kommandozeilenoptionen:

- `-i` zeigt nur die Prozesse an, die aktiv sind. Schlafende Prozesse werden ignoriert.
- `-b` (batch) kann in Kombination mit einem Redirektor verwendet werden, um die Ausgabe in eine Textdatei umzuleiten.
- `-d` (delay) gibt das Aktualisierungsintervall in Sekunden an. Der Standardwert ist eine Sekunde.
- `-q` startet `top` mit Echtzeitausgabe (wird nicht von allen Versionen des Programms unterstützt).

watch

Ein sehr nützlicher Helfer ist das Programm `watch`. Sie können es immer dann einsetzen, wenn ein anderes Programm eine Ausgabe auf dem Terminal macht und sich dann wieder beendet. Vielleicht wollen Sie z. B. Dateien dabei beobachten, wie diese ihre Größe ändern, während sie kopiert oder von `dd` erzeugt werden. Es gibt viele Situationen, in denen man mehrfach dasselbe Kommando gibt, um etwas zu beobachten.

Verwenden Sie das Kommando `watch ls -l` und Sie erhalten eine Ausgabe wie diese:

```
Alle 2,0s: ls -l                    arch-book: Sat Feb 24 16:14:00 2018

insgesamt 15977232
-rw-r--r-- 1 harald harald 3964551168  8. Jan 18:22 debian-9.3.0-amd64-DVD-1.iso
-rw-r--r-- 1 harald harald 4639948800 17. Feb 13:17 opensuse-leap-42.3-dvd-x86_64.iso
-rw-r--r-- 1 harald harald        159 24. Feb 15:39 SHA512SUMS
```

Beenden Sie das Programm mit Strg + c.

In der Voreinstellung führt watch das übergeben Kommando alle zwei Sekunden aus.

Wichtige Optionen für watch sind:

- -d hebt Änderungen zwischen zwei Updates hervor.
- -n gibt das Intervall an. Minimum ist 0.1 Sekunden.
- -t schaltet die Titelzeile ab.

Signale an Prozesse senden

Wenn Sie einen Prozess zur Laufzeit beeinflussen wollen, können Sie diesem Prozess ein Signal senden. Es gibt insgesamt 64 verschiedene Signale. Die gute Nachricht ist, dass Sie nicht alle Signale für die Prüfung kennen müssen. Sie erhalten eine Auflistung aller Signale, wenn Sie das folgende Kommando ausführen:

```
archangel:~ # kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
... weitere Zeilen wurden abgeschnitten ...
```

Die folgenden Signale werden häufig (auch manuell) an Prozesse übermittelt. Sie sind sowohl für die Praxis als auch für die Prüfung von Interesse:

- 1 bzw. SIGHUP ist das Hang-up-Signal, das ursprünglich Modems zum Auflegen veranlasste. Es wird auch benutzt, um Prozesse, die als Daemons laufen (bind, squid usw.), zu veranlassen, ihre Konfigurationsdateien neu einzulesen.
- 9 bzw. SIGKILL beendet einen Prozess mit Gewalt. Dieses Signal sollte nur in Notfällen verwendet werden, wenn Signal 15 kein Ergebnis bringt. Ein mit SIGKILL beendeter Prozess räumt nicht auf (temporäre Dateien usw.).
- 15 SIGTERM fordert einen Prozess auf, sich selbst zu beenden. Der Prozess erhält die Gelegenheit, Aufräumaktivitäten durchzuführen.

Weitere Signale, die häufig vorkommen, aber über Umwege an den Prozess gesendet werden, sind folgende:

- 2 bzw. SIGINT führt einen Programmabbruch aus. SIGINT wird gesendet, wenn die Tastenkombination Strg + C betätigt wird.
- 18 bzw. SIGCONT setzt einen Prozess fort, der zuvor mit SIGSTOP angehalten wurde. Das Signal SIGCONT wird durch die Kommandos fg und bg gesendet, die im nächsten Abschnitt behandelt werden.
- 19 bzw. SIGSTOP hält einen Prozess an. Der Prozess kann später mit SIGCONT fortgesetzt werden.

- 20 bzw. SIGTSTP hält einen Prozess an. Der Prozess bleibt speicherresident und kann mit SIGCONT fortgesetzt werden. Die Tastenkombination Strg + Z sendet dieses Signal.

kill

Wenn Sie ein Signal an einen Prozess senden wollen, verwenden Sie das Programm kill. Wie der Name schon vermuten lässt, ist kill primär dazu gedacht, ein Programm zu beenden. Der etwas morbide Begriff rührt wohl daher, dass man das Beenden eines Prozesses unter Unix auch »Sterben eines Prozesses« nennt. Das Kommando kill kann aber grundsätzlich alle 64 Signale (und nicht nur die »tödlichen«) an einen Prozess senden.

Die Syntax von kill ist sehr vielfältig, und Sie müssen für die Prüfung mit den verschiedenen Optionen vertraut sein. Als Argument erwartet kill eine oder mehrere durch Leerstellen getrennte *PIDs*. Ohne Optionen sendet kill das Signal 15 (SIGTERM) an den angegebenen Prozess. Wenn Sie einen Prozess mit kill beenden wollen, müssen Sie zunächst dessen *PID* ermitteln. Das können Sie mit dem Durchführen des bereits bekannten Programms ps erreichen. Im folgenden Beispiel soll ein außer Kontrolle geratener Midnight Commander beendet werden. Zunächst wird die *PID* ermittelt:

```
root@ubuntu-server:~# ps -au | grep mc
root  3167  0.2  0.4   5684  2268 pts/0  T  22:02 0:00 mc
root  3183  0.0  0.1   2884   752 pts/0  S+ 22:05 0:00 grep mc
```

Die *PID* ist demnach 3167. Die zweite Zeile, die grep ausgibt, ist das grep-Kommando mit der Suche nach dem mc selbst. Es soll zunächst ein SIGTERM (Signal 15) an den Prozess gesendet werden. Dazu stehen elf Varianten zur Verfügung:

```
root@ubuntu-server:~# kill 3167
root@ubuntu-server:~# kill -s 15 3167
root@ubuntu-server:~# kill -s SIGTERM 3167
root@ubuntu-server:~# kill -s TERM 3167
root@ubuntu-server:~# kill -s sigterm 3167
root@ubuntu-server:~# kill -s term 3167
root@ubuntu-server:~# kill -15 3167
root@ubuntu-server:~# kill -SIGTERM 3167
root@ubuntu-server:~# kill -TERM 3167
root@ubuntu-server:~# kill -sigterm 3167
root@ubuntu-server:~# kill -term 3167
```

Alle Kommandos führen zu demselben Ergebnis. Leider läuft der Midnight Commander immer noch. Das ist nicht überraschend, weil dieses Programm für dieses Verhalten bekannt ist. Das gilt übrigens auch für Netscape-Browser, die gerne noch Prozesse

aufrechterhalten, wenn X schon längst beendet wurde. Dieses Verhalten ist auch bei LPI bekannt. Ein solcher Prozess muss mit Gewalt beendet werden:

```
root@ubuntu-server:~# kill -SIGKILL 3167
```

Auch dieses Kommando kann in immerhin zehn Varianten ausgeführt werden. Die erste Variante von `SIGTERM` entfällt, weil `SIGTERM` die Voreinstellung ist.

In der Prüfung wird häufig folgendes Konstrukt verwendet:

```
root@ubuntu-server:~# kill -SIGHUP `cat /var/run/dhcpd.pid`
```

Mit dem Befehlsteil `cat /var/run/dhcpd.pid` wird die *PID* des `dhcpd` festgestellt. Die sogenannten *Backquotes* sorgen dafür, dass das Ergebnis dieses Kommandos an `kill -SIGHUP` angehängt wird. Eine andere Variante dieser Methode wäre:

```
root@ubuntu-server:~# kill -SIGHUP $(cat /var/run/dhcpd.pid)
```

killall

Damit Konstrukte wie das letzte Beispiel überflüssig werden, wurde das Programm `killall` entwickelt. Dieses Programm verwendet im Gegensatz zu `kill` keine *PIDs*, sondern Prozessnamen. Daraus ergeben sich zwei Vorteile:

- Die *PID* muss nicht ermittelt werden.
- Es können mehrere Prozesse mit gleichem Namen auf einmal beendet werden (deshalb heißt das Programm `killall`).

Ansonsten kann `killall` ähnlich wie `kill` verwendet werden. Das folgende Kommando beendet sofort alle Midnight-Commander-Instanzen mit Gewalt:

```
root@ubuntu-server:~# killall -s 9 mc
```

pgrep

Mit dem Programm `pgrep` können Sie die *PIDs* laufender Prozesse ermitteln. Sie müssen hierfür nicht einmal den vollständigen Namen eines Prozesses kennen. Es besteht außerdem die Möglichkeit, alle *PIDs* auszugeben, die einem bestimmten Benutzer zugeordnet sind, oder einfach nur festzustellen, wie viele Prozesse ein bestimmter Benutzer gerade verwendet. Beispiel:

```
root@archangel:~# pgrep -u harald -c
61
```

pkill

`pkill` ist der Syntax nach ähnlich wie `pgrep`. Dieses Programm beendet jedoch Prozesse anstatt lediglich die *PID* auszugeben. Das folgende Kommando beendet alle

Prozesse des Users »willi«. Das gilt auch für die Login-Shell, weshalb der Benutzer dann auch sofort abgemeldet wird:

```
root@archangel:~# pkill -u willi
```

Jobs im Vorder- und im Hintergrund

Sie können auf einer einzigen Shell praktisch beliebig viele Benutzerprozesse (Jobs) gleichzeitig laufen lassen. Es kann aber immer nur ein Prozess im Vordergrund laufen. Programme, die in den Hintergrund transferiert wurden, können den Status *running* oder *stopped* haben. Wenn sich ein Hintergrundprozess im Status *running* befindet, verrichtet er weiterhin seine Tätigkeit. Im Status *stopped* wartet er untätig im Hintergrund, bis er fortgesetzt oder endgültig beendet wird.

Wenn Sie ein Programm starten und anschließend die Tastenkombination Strg + Z ausführen, dann tritt das gestartete Programm in den Hintergrund. Außerdem sendet die Tastenkombination Strg + Z das Signal 20 (`SIGTSTP`) an den Prozess, so dass dieser vorübergehend gestoppt wird.

```
archangel:~ # updatedb
```

Hier wurde anschließend (nicht sichtbar) Strg + Z durchgeführt.

```
[1]+  Stopped                 updatedb
```

Normalerweise wird man bei einem Job wie `updatedb` wollen, dass dieser im Hintergrund arbeitet und seine Datenbank aktualisiert. In einem solchen Fall können Sie an das Kommando einfach ein Et-Zeichen (&) anhängen. Das Programm wird in den Hintergrund transferiert, bekommt aber kein `SIGTSTP`-Signal übermittelt. Deshalb wird dieser Prozess, im Hintergrund laufend, seine Arbeit verrichten. Beispiel:

```
archangel:~ # updatedb &
[1] 15749
```

Das Ergebnis des Kommandos kann sofort mit dem Kommando `jobs` überprüft werden. Sie sollten sich jedoch nicht zu lange Zeit lassen, weil der Job sonst bereits mit seiner Arbeit fertig ist:

```
archangel:~ # jobs
[1]+  Running                 updatedb &
```

Ist ein Job erledigt, bekommt er den Status *done*. Nach einmaliger Anzeige wird der Job nicht weiterhin gelistet:

```
archangel:~ # jobs
[1]+  Done                    updatedb
```

Wenn mehrere Prozesse in den Hintergrund transferiert wurden, liefert `jobs` eine Ausgabe wie die folgende:

```
archangel:~ # jobs
[1]-  Stopped                 updatedb
[2]+  Stopped                 /etc/cron.daily/do_mandb
```

Die Zahlen in den eckigen Klammern `[1]` sind Jobnummern, über die der jeweilige Job ansprechbar ist. Das Pluszeichen `+` markiert den Prozess, der von den Programmen `fg` und `bg` angesprochen wird, wenn explizit kein Job auf der Kommandozeile angegeben wurde.

Das Kommando jobs

Wie das Kommando `jobs` funktioniert, haben Sie in den vorangegangenen Beispielen bereits mehrfach gesehen. Dem ist eigentlich nur noch hinzuzufügen, dass die Option `-l` dafür sorgt, dass auch die *PIDs* der Prozesse angezeigt werden und nicht nur deren Jobnummern. Wenn Sie die Option `-p` verwenden, wird ausschließlich die *PID* angezeigt. Beispiel:

```
archangel:~ # jobs -l
[1]- 14746 Stopped                 updatedb
[2]+ 15395 Stopped                 /etc/cron.daily/do_mandb
```

Die *PIDs* sind allerdings für die Steuerung von Hintergrundprozessen im Wesentlichen nicht von Belang.

Die Kommandos bg und fg

Mit dem Kommando `bg` und der durch das Kommando `jobs` ermittelten Jobnummer können Sie einen gestoppten Hintergrundprozess im Hintergrund fortsetzen. Es wird dann ein `SIGCONT` (Signal 18) an den Prozess gesendet. Wenn Sie `bg` ohne Optionen ausführen, wird `SIGCONT` an den Prozess gesendet, dem in der Auflistung von `jobs` das Pluszeichen zugeordnet ist. Beispiel:

```
archangel:~ # bg 1
[1]- updatedb &
```

Anschließend wird mit `jobs` geprüft, ob das Kommando erfolgreich war:

```
archangel:~ # jobs
[1]-  Running                 updatedb &
[2]+  Stopped                 /etc/cron.daily/do_mandb
```

Wenn Sie einen Prozess, der sich im Hintergrund befindet, wieder in den Vordergrund holen möchten, verwenden Sie das Kommando `fg`, gefolgt von der Jobnummer. Es wird dann das Signal 18 (`SIGCONT`) an den Prozess gesendet. Wenn Sie `fg` keine Jobnummer angeben, wird der Prozess in den Vordergrund geholt und gegebenenfalls gestartet, dem in der Auflistung von `jobs` das Pluszeichen vorangestellt ist. Mit folgendem Kommando würde der Job `updatedb` aus dem vorangegangenen Beispiel in den Vordergrund geholt werden:

```
archangel:~ # fg 1
```


Prozesse unabhängig von einem Terminal laufen lassen

Wenn Sie einen Job unabhängig von einer Shell verwenden wollen, müssen Sie das Kommando `nohup` verwenden. Das kann sehr nützlich sein, wenn Sie z. B. kurz vor Feierabend einen Prozess in Gang bringen wollen, der lange Zeit in Anspruch nimmt. Das Kommando `nohup` macht einen Prozess immun gegen Signale wie `SIGHUB` und leitet außerdem *stdout* in eine Datei um, damit Programme, die Datenströme produzieren, zufriedengestellt werden. Diese Datei heißt *nohup.out* und wird im Heimatverzeichnis des Benutzers abgelegt. Beispiel:

```
archangel:~ # nohup updatedb &
archangel:~ # logout
```

Der Job läuft jetzt ohne Shell weiter.

screen

Wenn Sie ein System aus der Ferne administrieren, kann es in bestimmten Situationen gefährlich werden, wenn mitten in einer Aktion die Internetverbindung unterbrochen und deshalb ein laufendes Programm terminiert wird. Das gilt zum Beispiel für ein umfangreiches Systemupdate. Sie können solche administrativen Eingriffe relativ gefahrlos durchführen, wenn Sie ein Programm wie `screen` oder `tmux` verwenden. Beide Programme sind in der Lage, eine Sitzung aufrechtzuerhalten, auch wenn die Netzwerkverbindung unterbrochen wird. Am besten sehen Sie, wie es funktioniert, indem Sie folgendes Experiment durchführen:

- Starten Sie eine SSH-Sitzung zu einem entfernten Host.
- Geben Sie einfach nur das Kommando `screen` ein.
- Sie erhalten wieder eine scheinbar gewöhnliche Shell.
- Starten Sie einen Prozess, der aktiv etwas tut (z. B. `top`).
- Sorgen Sie anschließend dafür, dass die Verbindung abbricht.
- Verbinden Sie sich erneut mit dem entfernten System und geben Sie anschließend das Kommando `screen -r` ein.
- Sie sehen nun wieder das zuvor gestartete Programm `top`.

Das Programm `screen` versteht etliche Optionen, die Sie wie immer in der entsprechenden Manpage wiederfinden. Eine weitere Funktionalität von `screen` sind mehrere Terminals in einer einzigen SSH-Sitzung.

tmux

Der Name `tmux` steht für Terminalmultiplexer. Das Programm bietet ähnliche Funktionen wie `screen`, verwendet aber auf die *ncurses*-Bibliothek. `tmux` basiert außerdem auf einer Client-Server-Architektur und bietet viele zusätzliche Funktionen, die über Tastenkombinationen aufgerufen werden. Sie erhalten einen Überblick, wenn Sie innerhalb einer `tmux`-Sitzung Strg + B gefolgt von einem ? eingeben.

[+]

Praxistipp

Das Programm `byobu`, das übrigens nicht prüfungsrelevant ist, leistet ähnliche Dienste, ist aber erheblich angenehmer zu bedienen, weil es die Vorzüge von `tmux` und `screen` miteinander vereint. Ich kann nur jedem Administrator empfehlen, es einmal auszuprobieren.

Das Programm uptime

Dieses Programm gibt die aktuelle Zeit, die Laufzeit des Systems, die Anzahl der Benutzer und die durchschnittliche Anzahl von lauffähigen Jobs in den letzten 1, 5 und 15 Minuten aus.

Hier sehen Sie die Ausgabe des Kommandos `uptime`. Die Bedeutung der Ausgabe des Befehls wird im Anschluss beschrieben:

```
[root@centos ~]# uptime
11:07:44 up 284 days,19:05,2 user,load average: 0.37, 0.11, 0.03
```

Als Erstes wird die aktuelle Systemzeit (11:07:44) ausgegeben. Das System läuft demnach bereits seit 284 Tagen, 19 Stunden und 5 Minuten. Es sind aktuell zwei Benutzer verbunden. Die folgenden drei Werte zeigen an, wie viele Jobs durchschnittlich in den letzten 1, 5 und 15 Minuten gelaufen sind.

Wenn Sie das Kommando `w` verwenden, um die aktuell angemeldeten Benutzer anzuzeigen, wird in der ersten Zeile ebenfalls die *uptime* angezeigt:

```
[root@centos ~]# w
11:07:44 up 284 days,19:05,2 user,load average: 0.70, 0.11, 0.07
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
harald   tty1     -              28Mar09 20days  0.02s  0.02s -bash
root     pts/0    192.168.50.200   10:34    0.00s  0.38s  0.00s w
```


Informationen über angemeldete Benutzer beziehen übrigens beide Programme aus der Datei */var/run/utmp*.

Das Kommando free

Um festzustellen, wie die momentane Speicherauslastung eines Systems aussieht, können Sie das Kommando `free` verwenden. Die Ausgabe des Programms ist recht breit, weshalb sie sich in einem Buch nicht vernünftig darstellen lässt. Das Programm kennt folgende Optionen, die auch in der Prüfung abgefragt werden könnten:

- `-b` (bytes) Ausgabe in Bytes
- `-k` (kilo) Ausgabe in KB
- `-m` (mega) Ausgabe in MB
- `-g` (giga) Ausgabe in GB
- `-l` Details aus High- und Low-Memory anzeigen
- `-o` (old) altes Format verwenden
- `-t` (total) zeigt Summen für RAM und swap
- `-s` (seconds) automatische Aktualisierung nach der angegebenen Zeit in Sekunden (z. B. `free -s 5` für alle 5 Sekunden)
- `-c` (count) Häufigkeit der mit `-s` gewünschten automatischen Aktualisierung
- `-V` (Version) das ist nicht *verbose*, wie man denken könnte!

103.6 Prozess-Ausführungsprioritäten ändern

Wichtung: 2

Beschreibung: Kandidaten sollten in der Lage sein, die Ausführungsprioritäten von Prozessen zu verwalten.

Wichtigste Wissensgebiete:

- die Standardpriorität eines neu erzeugten Jobs kennen
- ein Programm mit einer höheren oder niedrigeren Priorität als im Normalfall laufen lassen
- die Priorität eines laufenden Prozesses ändern

Liste wichtiger Dateien, Verzeichnisse und Anwendungen:

- `nice`
- `ps`
- `renice`
- `top`

Allgemeines

Manchmal ist es nötig, einen Prozess mit einer höheren oder niedrigeren Priorität als der normalen Priorität laufen zu lassen. So kann ein Prozess, dessen Ausführungsgeschwindigkeit nicht wichtig ist, mit einer niedrigeren Priorität laufen als ein systemkritischer Prozess. Sie können diese Prioritäten sowohl beim Start des Programms als auch nachträglich zur Laufzeit des Programms festlegen. Ein normaler Benutzer kann allerdings einem Prozess lediglich eine niedrigere Priorität zuordnen. Damit soll verhindert werden, dass Benutzer, die ihre eigene Arbeit als besonders wichtig einstufen, das ganze System lahmlegen.

nice

Mit dem Kommando `nice` legen Sie den Nice-Wert eines Programms schon beim Start fest. Sie sollten für die Prüfung unbedingt mit den unterschiedlichen Möglichkeiten dieses Kommando zu verwenden vertraut sein. Wenn Sie ein Programm mit `nice` ausführen, ohne Optionen anzugeben, wird dieses Programm um den Nice-Wert 10 »netter«. Es gibt Nice-Werte von –20 bis +19. Wenn Sie einen negativen Wert angeben, wird dem Programm eine höhere Priorität bei der Ausführung zugewiesen. Der Nice-Wert und die Prozesspriorität sind nicht dasselbe. Es wird durch `nice` lediglich Einfluss auf die Prozesspriorität genommen.

Bei der Übergabe von Optionen erwartet `nice` immer einen vorangestellten Bindestrich. Wenn ein Programm mit einem positiven Nice-Wert gestartet werden soll, stehen Ihnen zwei Möglichkeiten zur Ausführung des Kommandos zur Verfügung:

```
harald@archangel:~> nice -n 12 nano
harald@archangel:~> nice -12 nano
```

In beiden Fällen wird der Editor `nano` mit einem positiven Nice-Wert von 12 gestartet. Der Strich in der zweiten Zeile ist also kein Minus, sondern dient der Einleitung der Option. Wenn Sie einen negativen Nice-Wert verwenden wollen, um die Ausführungspriorität eines Programms zu erhöhen, können Sie folgende Kommandos verwenden:

```
michi@archangel:~ # nice -n -15 pico
michi@archangel:~ # nice --15 pico
```

Die zweite Zeile enthält jetzt zwei Striche. Der erste Strich leitet die Option ein, und der zweite ist diesmal ein Minuszeichen.

Prüfungstipp

Lassen Sie sich in der Prüfung nicht von dem doppelten Strich irritieren. Es handelt sich hier nicht um eine doppelte Verneinung.

renice

Wollen Sie die Ausführungspriorität eines Prozesses zur Laufzeit ändern, dann verwenden Sie `renice` oder `top`. Die Behandlung von Nice-Werten mit `top` wird etwas später in diesem Buch beschrieben.

Die Syntax weicht von der des `nice`-Kommandos ab, was leider oft für Verwirrung sorgt. Im Normalfall geben Sie `renice` ohne Bindestrich einfach den neuen Nice-Wert und die *PID* des zu ändernden Prozesses an. Das Programm teilt anschließend den alten und den neuen Nice-Wert mit:

```
archangel:~ # renice -10 5309
5309: old priority 12, new priority -10
```

Hier ist also der Strich ein Minus und keine Einleitung für eine Option. Merken Sie sich unbedingt diesen Unterschied zu `nice`.

Mit der Option `-u` (hier wiederum ist der Strich unabdingbar) können Sie alle Prozesse eines einzelnen Benutzers gleichzeitig beeinflussen. Die Syntax ist ansonsten mit der identisch, die man auch bei der Manipulation eines Prozesses benutzen würde:

```
archangel:~ # renice 5 -u dominik
1002: old priority 0, new priority 5
```

top und ps zur Überprüfung von Prioritäten

Ob ein Programm mit einer normalen Priorität läuft oder nicht, können Sie mit dem bereits bekannten Programm `ps` überprüfen:

```
archangel:~ # ps -au
...Zeilen wurden entfernt...
harald  2698  0.0  0.0  3024  1076 pts/2  SN+  11:45   0:00 nano
michi   2736  0.0  0.0  2368   940 pts/3  S<+  11:46   0:00 pico
dominik 3416  0.1  0.0  5616  2012 pts/4   S+  12:00   0:00 mc
... Zeilen wurden entfernt ...
```

Die achte Spalte in der Ausgabe des Kommandos ist u. a. für diese Information zuständig. Wenn ein Prozess `nice` ist, wird in dieser Spalte ein großes `N` auftauchen. Bei einem Prozess, der mit einer erhöhten Priorität läuft, finden Sie dort ein `<`. Die anderen Zeichen in dieser Spalte haben mit den Prioritäten nichts zu tun. Das große `S` steht für *interruptible sleep* und das Pluszeichen `+` weist auf einen Prozess hin, der im Vordergrund ausgeführt wird. Die Programme `pico` und `mc` wurden also offensichtlich nicht mit dem Kommando `nice` gestartet.

Um die tatsächliche Priorität eines Prozesses und den Wert für `nice` zu ermitteln, benötigen Sie das Programm `top`:

```
PID USER    PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
2736 michi   0 -15  2368  952  748 S  0.2  0.0   0:00.01 pico
2698 harald  27 12  3024 1084  892 S  0.2  0.0   0:00.02 nano
3416 dominik 16  0  5616 2028 1580 S  1.2  0.1   0:00.22 mc
```

Hier wird auch deutlich, dass eine Prozesspriorität und der Wert für `nice` nicht dasselbe sind. Der mit `nice` bzw. `renice` eingestellte Wert hat lediglich Einfluss auf die Prozesspriorität. Die dritte Spalte der Ausgabe von `top` entspricht der Priorität, die das System einem Prozess zugewiesen hat. Der Standardwert ist 16. In der vierten Spalte findet sich der Nice-Wert.

Sie können mithilfe von `top` auch einen Nice-Wert ändern. Dazu müssen Sie lediglich zur Laufzeit von `top` die Taste R betätigen. Sie werden dann nach der *PID* des zu ändernden Prozesses gefragt. Anschließend müssen Sie den neuen Nice-Wert angeben.

103.7 Textdateien mit regulären Ausdrücken durchsuchen

Wichtung: 3

Beschreibung: Kandidaten sollten in der Lage sein, Dateien und Textdaten mit regulären Ausdrücken zu manipulieren. Dieses Prüfungsziel umfasst etwa die Erstellung einfacher regulärer Ausdrücke, die mehrere Beschreibungselemente enthalten. Es umfasst ebenfalls den Einsatz von Werkzeugen, die reguläre Ausdrücke zum Durchsuchen eines Dateisystems oder von Dateiinhalten verwenden.

Wichtigste Wissensgebiete:

- einfache reguläre Ausdrücke mit mehreren Beschreibungselementen erstellen
- den Unterschied zwischen grundlegenden und erweiterten regulären Ausdrücken verstehen
- Verstehen der Konzepte von Sonderzeichen, Zeichenklassen, Quantifizierern und Ankern
- Werkzeuge verwenden, die mit regulären Ausdrücken Dateisysteme oder Dateiinhalte durchsuchen
- reguläre Ausdrücke verwenden um Text zu löschen, ändern oder zu substituieren und arbeiten mit Bereichen

Liste wichtiger Dateien, Verzeichnisse und Anwendungen:

- `grep`
- `egrep`
- `fgrep`

- `sed`
- *regex*(7)

Allgemeines

In den nächsten Abschnitten geht es um die Verwendung regulärer Ausdrücke. Sie werden in diesem Zusammenhang auch in der Prüfung wahrscheinlich häufiger auf den englischen Originalbegriff *regular expression* oder die Kunstwörter *regexp* und *regex* stoßen. Gemeint ist damit immer dasselbe. Auch wenn viele andere Programme ebenfalls Gebrauch von regulären Ausdrücken machen, werden Sie in der Prüfung ausschließlich in Verbindung mit dem Stream-Editor `sed` oder dem Programm `grep` verwendet. Andere Programme, die *regexp* verwenden, sind z. B. *Python*, *awk*, *gawk* oder *Perl*. Dem Editor `vi`, der natürlich auch *regex* verwendet, ist ein eigenes Kapitel gewidmet.

Reguläre Ausdrücke

Ein regulärer Ausdruck kann als eine Art Beschreibungssprache verstanden werden. Hierbei werden Suchmuster verwendet, wie z. B. ein Suchwort, eine Position im Text (Zeilenanfang, Zeilenende) oder auch Wildcards. Es wäre schön, wenn reguläre Ausdrücke von allen Programmen gleich interpretiert würden. Leider ist das aber nicht so.

Prüfungstipp

Es ist empfehlenswert, sich zunächst mit den Bestandteilen regulärer Ausdrücke zu befassen, die von den prüfungsrelevanten Programmen (`sed`, `grep`, `vi`) gemeinsam genutzt werden.

[+]

Allgemeingültige Metazeichen

- ^ Textanker für den Zeilenanfang
- $ Textanker für das Zeilenende
- \< markiert einen Wortanfang
- \> markiert ein Wortende
- \ schützt ein Zeichen vor der Interpretation. Das kann notwendig werden, wenn Sie etwa nach einem Dollarzeichen suchen müssen. Dieses würde ohne den vorangestellten Backslash als Zeilenende interpretiert werden.
- [] – Der Ausdruck passt auf die enthaltenen Zeichen. `[abcd]` ist demnach zutreffend, wenn der Ausdruck ein a, b, c oder d enthält. Derselbe Ausdruck hätte auch mit `[a-d]` beschrieben werden können.

- `[^Zeichen]` – Der Ausdruck trifft nicht auf »Zeichen« zu. Hier besteht (auch und insbesondere in der Prüfung) Verwechslungsgefahr mit dem Zeilenanfang.

[+]

Prüfungstipp

Die Zeichenkette `^[^#]` erzielt einen Treffer, wenn am Zeilenanfang keine Raute (#) steht. Der erste Zirkumflex bezeichnet den Zeilenanfang, während der zweite das »Nicht« für die Raute darstellt. Denken Sie daran: Das ist ein sehr beliebtes Prüfungsthema.

Wildcards (Joker)

Wildcards ersetzen einzelne Zeichen innerhalb eines Suchmusters.

- `*` bezeichnet null oder beliebig viele Wiederholungen des voranstehenden Zeichens. Das ist bei Ersetzungen sinnvoller, als es zunächst scheint.
- `?` bezeichnet null oder eine einmalige Wiederholung des voranstehenden Zeichens. Dieser Joker wurde von `sed` und `grep` ursprünglich nicht unterstützt. In diesem Fall können Sie diesen Joker aber mit `egrep` verwenden. Neuere Versionen von `grep` unterstützen auch das Fragezeichen.
- `+` bezeichnet das zumindest einmalige Vorkommen des voranstehenden Zeichens.
- `.` (Punkt) kann als Platzhalter für ein beliebiges Zeichen (außer Zeilenvorschub) verwendet werden.

[+]

Prüfungstipp

Achtung: In der Shell ersetzt der Stern (*) mehrere beliebige Zeichen und das Fragezeichen (?) ein beliebiges Zeichen. Achten Sie in der Prüfung darauf, dies nicht zu verwechseln. Beispiele:

```
archangel:/bin # ls g*
gawk  getkeycodes  grep  guessfstype  gunzip  gzip
archangel:/bin # ls g???
gawk  grep  gzip
```

Die Verwendung von grep

Mit dem Kommando `grep` können Sie die Zeilen einer Datei nach regulären Ausdrücken durchsuchen. Die gefundenen Zeilen werden nach *stdout* ausgegeben. Der wesentlichste Unterschied zu `sed` ist, dass `grep` keine Änderungen an den Inhalten einer Datei vornimmt. Die Bedeutung von `grep` ist wohl sinngemäß ***G**lobal search for **R**egular **E**xpressions and **P**rint out*. Mittlerweile existieren allerdings sowohl im Inter-

net als auch in der einschlägigen Fachliteratur viele leicht abweichende Interpretationen dieses Begriffes.

Bevor es an die praktische Arbeit mit `grep` und den regulären Ausdrücken geht, sollten Sie noch die wichtigsten Optionen von `grep` kennen:

- `-v` invertiert die Ausgabe. Es werden also die Zeilen ausgegeben, auf die der angegebene reguläre Ausdruck nicht zutrifft. Das ist kein Verbose-Mode.
- `-n` fügt dem Suchergebnis eine Zeilennummerierung hinzu.
- `-E` aktiviert die Verwendung von erweiterten *regex*. Das Programm verhält sich dann wie `egrep`.
- `-c` zählt lediglich die Anzahl der Übereinstimmungen mit dem Suchmuster.
- `-i` ignoriert die Groß-/Kleinschreibung.

Beispiele zur Verwendung von grep

```
archangel:~ # grep Willi Adressliste
```

Sucht in der Datei *Adressliste* Zeilen, die das Suchwort »Willi« enthalten.

```
archangel:~ # grep [Ww]ill[iy] Adressliste
```

Sucht in der Datei *Adressliste* nach »Willi«, »Willy«, »willi« und »willy«. Die Zeichen in den eckigen Klammern verstehen sich hier also als Aufzählung. Jede Klammergruppe ersetzt genau ein Zeichen. Ein ähnliches Ergebnis liefert das folgende Kommando:

```
archangel:~ # grep -i will[iy] Adressliste
```

Mit der Option `-i` ignoriert `grep` die Groß-/Kleinschreibung. Allerdings gilt das dann für alle Zeichen innerhalb der Zeichenkette. Es würde demzufolge z. B. auch der Name »wILLy« gefunden werden.

Im nächsten Beispiel wird das zweite Zeichen durch einen Punkt ersetzt. Ein Punkt repräsentiert genau ein beliebiges Zeichen (außer Zeilenvorschub).

```
archangel:~ # grep W.lli Adressliste
```

Als Ergebnis würde in der Adressliste auch »Wolli« oder »Wqlli« (falls dieser Name überhaupt existiert) gefunden werden.

Wenn Sie Konfigurationsdateien untersuchen wollen, kann es passieren, dass Ihnen die umfangreiche Dokumentation innerhalb dieser Dateien oder überflüssige Leerzeilen die Übersicht erschweren. Das ist ein klassisches Einsatzgebiet für `grep` in Kombination mit Textankern. Es wird in den folgenden Beispielen davon ausgegangen, dass die fragliche Konfigurationsdatei mit Rauten (Hashes) am Zeilenanfang als Kommentarzeichen arbeitet, wie es unter Linux eigentlich fast immer der Fall ist. Die

folgenden Beispiele sind so ausgewählt, dass prüfungsnahe Kommandos zum Einsatz kommen:

```
archangel:~ # grep [#] /etc/config.conf
```

Hier werden alle Zeilen ausgegeben, die eine Raute (#) enthalten. Das ist aber nicht das gewünschte Ergebnis, sondern viel eher das Gegenteil davon. Im nächsten Schritt soll das Ergebnis invertiert werden:

```
archangel:~ # grep -v [#] /etc/config.conf
archangel:~ # grep [^#] /etc/config.conf
```

Die erste Variante invertiert einfach mit der Option `-v` die Ausgabe von `grep`. In der zweiten Zeile wird der Raute ein »Nicht« vorangestellt. Dieses »Nicht« muss unbedingt innerhalb der Klammern gesetzt werden, weil der Inhalt der Klammern genau einem Zeichen entspricht. Würde das Zeichen vor die Klammer gestellt, würde es als Zeilenanfang interpretiert werden. Das Ergebnis wäre dann die Ausgabe aller Zeilen, die eine Raute am Zeilenanfang enthalten. Leider findet das zweite Kommando alle Zeilen, die Zeichen enthalten, die keine Rauten sind. Da das überhaupt nicht das erwünschte Ergebnis ist, muss das Kommando wohl noch modifiziert werden, wie Sie gleich noch sehen werden.

Normalerweise stehen Kommentarzeichen am Anfang einer Zeile. Damit erwünschte Rauten, die inmitten der Konfigurationsdatei auftauchen, nicht mit ausgesiebt werden, muss die Position des Zeichens noch festgelegt werden:

```
archangel:~ # grep -v ^[#] /etc/config.conf
archangel:~ # grep ^[^#] /etc/config.conf
```

Der Zirkumflex steht für den Zeilenanfang. In der zweiten Version repräsentiert der erste Zirkumflex den Zeilenanfang und der Zirkumflex innerhalb der Klammern das »Nicht«. Leider enthält das Ergebnis noch Leerzeilen. Diese werden im nächsten Kommando mit einem zweiten `grep`-Kommando entfernt. Eine Leerzeile besteht aus einem Zeichen für den Zeilenanfang (^) und einem Zeichen für das Zeilenende ($). Wenn diese beiden Zeichen aufeinander folgen, kann es sich also nur um eine Leerzeile handeln. `grep` soll nach diesen Zeilen suchen und die Ausgabe invertieren:

```
archangel:~ # grep -v ^[#] /etc/config.conf | grep -v ^$
archangel:~ # grep ^[^#] /etc/config.conf | grep -v ^$
```

Es folgen nun einige Beispiele, die das Verständnis für die mit `grep` verwendbaren regulären Ausdrücke fördern sollen.

Das folgende Kommando gibt Zeilen aus, die entweder die Zeichenkette »Wort« allein oder »Wort«, gefolgt von einer beliebigen Anzahl von Ziffern zwischen 1 und 9, enthalten (Wort, Wort1, Wort47, Wort6279 usw.):


```
archangel:~ # grep 'Wort[1-9]*\>' Suchdatei
```

Das Zeichen > (größer als) ist ein Metazeichen und repräsentiert das Ende des Wortes. In diesem Fall muss das Zeichen > durch den Backslash vor `grep` geschützt werden. `grep` würde sonst nur die Zeilen ausgeben, die am Ende ein > aufweisen. Leicht abweichend vom vorangegangenen Beispiel sollen jetzt nur die Zeilen ausgegeben werden, die das Suchmuster »Wort« enthalten oder »Wort«, gefolgt von einer einzelnen Ziffer.

```
archangel:~ # grep 'Wort[1-9]\?\>' Suchdatei
```

Beachten Sie, dass sowohl das ? als auch das > durch einen Backslash geschützt werden müssen.

Um alle Einträge aus der Datei */var/log/syslog* anzuzeigen, die beispielsweise am 10.12. vorgenommen wurden, können Sie folgendes Kommando verwenden:

```
archangel:~ # grep '^Dec 10' /var/log/syslog
```

Mit dem folgenden Kommando werden alle Zeilen einer Datei angezeigt, die drei, vier oder fünf aufeinander folgende Einsen enthalten:

```
archangel:~ # grep -w '1\{3,5\}' Suchdatei
```

In diesem Fall müssen die geschweiften Klammern jeweils mit einem Backslash geschützt werden. Ohne diesen »Schutz« würde `grep` nach dem Muster `1{3,5}` in der Datei suchen. Diese Konstellation kann verwendet werden, um die Anzahl der Wiederholungen eines einzelnen Zeichens genau festzulegen. Wildcards grenzen die Anzahl der Zeichenwiederholung nur grob ein. Wenn Sie lediglich eine Mindestanzahl von Wiederholungen ohne Limit nach oben angeben wollen, dann verwenden Sie folgendes Konstrukt:

```
archangel:~ # grep -w '1\{3,\}' Suchdatei
```

Es werden hiermit alle Zeilen ausgegeben, die mindestens drei aufeinander folgende Einsen aufweisen. Die folgende Befehlszeile sucht nach Zeilen, die drei aufeinander folgende Ziffern beinhalten:

```
archangel:~ # grep -w '[0-9]\{3\}' Suchdatei
```

Wie bereits beschrieben, repräsentieren Angaben in rechteckigen Klammern immer genau ein Zeichen, in diesem Fall die Ziffern von 0 bis 9. Das so beschriebene Zeichen soll dreimal vorkommen. Die Angabe `\{3\}` legt genau das fest. Sollen stattdessen drei aufeinander folgende Großbuchstaben in einer Zeile enthalten sein, wird folgendes Kommando diese Zeilen auffinden:

```
archangel:~ # grep -w '[A-Z]\{3\}' Suchdatei
```

Denken Sie daran, dass einige Metazeichen durch einen Backslash geschützt werden müssen, damit grep diese nicht als Suchmusterbestandteil versteht. Diese Zeichen sind: ?, +, {, }, |, (und). Umgekehrt kann es aber auch vorkommen, dass nach einem Zeichen gesucht werden soll, das grep als Metazeichen verstehen würde. Dann muss auch dieses Zeichen durch einen Backslash geschützt werden, aber aus dem gegenteiligen Grund. Beispiel:

```
archangel:~ # grep '\$ [0-9]\{3,\}\>' Suchdatei
```

Dieses Kommando sucht nach mindestens dreistelligen Dollarbeträgen in einer Datei. Der erste Backslash sorgt dafür, dass das Dollarzeichen nicht als Metazeichen für das Zeilenende interpretiert wird. Die drei anderen Backslashes schützen die geschweiften Klammern und das »Größer als«-Zeichen davor, als Bestandteil des Suchmusters interpretiert zu werden.

Das letzte Beispiel sucht nach Zeilen, die zumindest einen Großbuchstaben enthalten. Auch hier muss der Joker durch einen Backslash geschützt werden:

```
archangel:~ # grep '[A-Z]\+' Suchdatei
```

[!]

Achtung

Je nach verwendeter Version von grep kann es zu unterschiedlichem Verhalten kommen. Es gibt z. T. sogar Abweichungen zwischen Manpages und tatsächlichem Verhalten. Lassen Sie sich davon bitte nicht zu lange aufhalten.

egrep und fgrep

Die beiden Programme egrep(extended) und fgrep (fixed) waren ursprünglich Varianten von grep mit zusätzlichen Fähigkeiten. Beide sind heutzutage eigentlich nicht mehr nötig, weil grep mit den entsprechenden Optionen dieselben Aufgaben ausführen kann wie egrep und fgrep.

- grep -E entspricht egrep.
- grep -F entspricht fgrep.

Weil ältere Skripte eventuell noch egrep oder fgrep verwenden, werden diese beiden Programme normalerweise weiterhin in Linux-Distributionen gepflegt.

egrep unterstützt einen erweiterten Satz Metazeichen gegenüber grep. Entsprechend wird bei der Suche auch mehr Rechenzeit benötigt. Das genaue Gegenteil macht fgrep. Wenn in der zu suchenden Zeichenkette Metazeichen vorkommen, werden diese als normale Zeichen interpretiert.

Weitere Verwandte von grep sind agrep für fehlertolerantes Suchen und zgrep für die Suche in komprimierten Archiven. Diese sind aber bei der Prüfung nicht von Belang.

Die Verwendung von sed

sed ist ein Stream-Editor. Das bedeutet, dass es sich nicht um einen interaktiven Editor handelt, wie z. B. vi, nano oder pico. Er liest, wenn keine Datei angegeben wird, von der Standardeingabe und gibt das Ergebnis an *stdout* wieder aus. Deshalb werden Sie die Ausgabe von sed in eine Datei umleiten müssen, wenn Sie die Änderungen abspeichern wollen. Hierbei darf die Zieldatei nicht mit der Quelldatei identisch sein. Bei der Ausführung des Kommandos mit einem einfachen Redirektor würde die Shell die Datei zuerst löschen und dann neu anlegen. Das Ergebnis wäre eine leere Datei. Beispiel:

```
archangel:~ # sed 's/rumba/samba/g' Suchdatei1 >Suchdatei1
```

Dieses Kommando hat dann, wie gesagt, eine leere Datei zur Folge. Sie können dieses Problem allerdings umgehen, indem Sie die Option -i (bzw. --in-place) verwenden. Das entsprechende Kommando würde dann so aussehen:

```
archangel:~ # sed -i 's/rumba/samba/g' Suchdatei1
```

Bei der Arbeit mit sed verwendet man Optionen und Kommandos, wobei die Optionen allerdings meist eine eher untergeordnete Rolle spielen. Wenn keine Optionen angegeben werden, kann sed allerdings auch nur ein Kommando mit einer einzigen Befehlszeile entgegennehmen.

Optionen

- -e Kommando – Diese Option ist, wenn nur ein einziges Kommando verwendet wird, optional. Sollen mehrere Kommandos in einer Befehlszeile verwendet werden, muss diese Option jedem Kommando vorangestellt werden.
- -f Scriptdatei – Mit dieser Option wird eine Skriptdatei angegeben, welche die Kommandos für sed enthält. Diese Methode ist besonders nützlich, wenn mehrere sed-Kommandos routinemäßig ausgeführt werden müssen.
- -g – Im Normalfall reagiert sed nur auf das erste Vorkommen eines Suchmusters innerhalb einer Zeile. Wenn die Option -g gesetzt wird, gelten alle folgenden Kommandos als global. Es ist aber auch möglich, eine globale Verfahrensweise in ein Kommando aufzunehmen.

Adressierung

Standardmäßig bearbeitet sed alle Zeilen der angegebenen Datei(en). Es kann aber vorkommen, dass dieses Verhalten nicht erwünscht ist. Dann müssen Sie dem Editor mitteilen, an welcher Textposition er seine Arbeit beginnen bzw. aufhören soll. Für

diese Positionsangaben verwenden Sie Adressen. Sie können eine, zwei oder gar keine Adresse verwenden. Es gibt grundsätzlich nur drei Arten von Adressen für sed:

- Zeilennummern
- das Dollarzeichen ($) für die letzte Zeile
- ein regulärer Ausdruck, der auf beiden Seiten durch Slashes begrenzt wird (z. B. /regexp/)

Wenn sed eine Datei komplett verarbeiten soll, geben Sie einfach gar keine Adresse ein. Sollen z. B. in einem bestimmten Bereich eines Textes Ersetzungen vorgenommen werden, dann geben Sie zwei Adressen an. Diese sind dann durch ein Komma voneinander zu trennen. Einige Kommandos arbeiten mit nur einer Adresse. So hängt das Kommando a (append) an einer Zeile Text an, auf die die Adresse passt. Eine *Von-bis*-Angabe hätte hier wenig Sinn.

Kommandos

Es gibt zwei Methoden, Kommandos an sed zu übergeben. Entweder Sie geben die Kommandos direkt auf der Kommandozeile ein, oder Sie verwenden ein Skript. Erfreulicherweise ist die Syntax für die Kommandos in beiden Fällen annähernd identisch. Am besten lernen Sie diese Kommandos durch eigenes Anwenden. Deshalb werden sie im Folgenden anhand einiger Beispiele demonstriert.

Das erste Beispiel zeigt das Kommando y. Dieses Kommando wird für die Übersetzungen von einem Zeichen in ein anderes Zeichen verwendet. Es können auch mehrere Zeichen übersetzt werden, aber z. B. niemals ein Zeichen in drei andere Zeichen. Es besteht immer eine 1:1-Beziehung, ähnlich wie bei dem Kommando tr. Das folgende Beispiel veranschaulicht die Konvertierung einer kommaseparierten Datei (CSV) in eine semikolonseparierte Datei:

```
archangel:~ # sed 'y/,/;/' Kommadatei > Semikolondatei
```

Das Kommando d (delete) löscht ganze Zeilen, wenn ein regulärer Ausdruck zutrifft, oder auch Zeilen, die mit Adressen angegeben wurden:

```
archangel:~ # sed -e '/^$/d' -e '/^#/d' /etc/config.conf
```

Mit dem voranstehenden Befehl wurden sowohl Kommentarzeilen (Hash am Zeilenanfang ^#) als auch Leerzeilen (Aufeinanderfolge von Zeilenanfang ^ und Zeilenende $) aus einer Konfigurationsdatei entfernt. Da hier zwei Kommandos gleichzeitig an sed übergeben wurden, musste diesmal der ansonsten optionale Schalter -e zweimal verwendet werden.

Das nächste Beispiel zeigt die Löschung von Zeilen, basierend auf Adressen:

```
archangel:~ # sed '25,47d' Originaldatei > Kurzdatei
```


Bei der Zeichenkette 25,47 handelt es sich um die kommagetrennten Adressen, hier also Zeilennummern. Das Ergebnis des Kommandos ist die Ausgabe aller Zeilen von 1 bis 24 und der Zeilen 48 bis zum Ende.

Sie können die Ausgabe des Kommandos auch einfach invertieren, indem Sie zusätzlich ein Ausrufezeichen verwenden:

```
archangel:~ # sed '25,47!d' Originaldatei > Kurzdatei
```

Als Ergebnis werden die Zeilen 25 bis 47 ausgegeben. Alle anderen Zeilen werden gelöscht.

Mit der Substitution s können Zeichen übersetzt werden. Im Vergleich zu y ist dieses Kommando leistungsfähiger, weil es auch mehrere Zeichen in ein einziges Zeichen überführen kann und umgekehrt:

```
archangel:/textfiles # sed 's/ue/ü/' Namensliste
```

Mit diesem Befehl wird das erste in einer Zeile gefundene ue in ein ü übersetzt. Der Befehl kann durch das Hinzufügen des sed-Kommandos g globalisiert werden. Dadurch wird sichergestellt, dass sed auch bei mehrfachem Auftreten der Zeichenkette ue innerhalb einer Zeile diese ebenfalls in ein ü übersetzt:

```
archangel:/textfiles # sed 's/ue/ü/g' Namensliste
```

Wenn Sie eine ganze Serie von Kommandos an sed übergeben wollen, empfiehlt sich die Verwendung einer Skriptdatei. Es ist zu beachten, dass die Kommandos innerhalb der Skriptdatei einzeln und zudem nacheinander abgearbeitet werden. Wenn das erste Kommando einige Zeilen löscht und das zweite Kommando Zeilennummern als Adressen verwendet, dann sind diese Zeilennummern durch die Ausführung des ersten Kommandos verschoben. Dieser Umstand muss bei der Planung eines Skripts berücksichtigt werden.

Das letzte Beispiel demonstriert, wie sed mit einer Skriptdatei verwendet wird, um die Umlaute einer Textdatei in ein HTML-taugliches Format zu konvertieren. Die Meta-Tags für HTML müssen nachträglich eingefügt werden. Zunächst wird die Skriptdatei mit folgendem Inhalt erstellt:

```
archangel:~ # cat sed-umltohtml-script
s/ä/\&auml\;/g;
s/Ä/\&Auml\;/g;
s/ü/\&uuml\;/g;
s/Ü/\&Uuml\;/g;
s/ö/\&ouml\;/g;
s/Ö/\&Ouml\;/g;
s/ß/\&szlig\;/g;
```

Danach wird diese Serie von Kommandos mit `sed -f` auf die Quelldatei (*umltext*) angewendet und das Ergebnis in eine neue Textdatei (*htmltext*) umgeleitet. Der Backslash schützt jeweils das Et-Zeichen vor der Interpretation. Aus `ä` wird dann beispielsweise `&auml;`:

```
archangel:~ # sed -f sed-umltohtml-script umltext > htmltext
```

Das war ein sehr einfaches Beispiel. Sie können mit `sed` noch wesentlich komplexere Arbeiten ausführen. Für die Prüfung jedoch ist das hier Beschriebene mehr als ausreichend.

103.8 Grundlegendes Editieren von Dateien mit dem vi

Wichtung: 3

Beschreibung: Kandidaten sollten in der Lage sein, Textdateien mit dem *vi* zu editieren. Dieses Prüfungsziel umfasst *vi*-Navigation, grundlegende *vi*-Modi, Einfügen, Ändern, Löschen, Kopieren und Finden von Text.

Wichtigste Wissensgebiete:

- mit *vi* in einem Dokument navigieren
- verstehen und verwenden der *vi*-Modi
- in *vi* Text einfügen, ändern, löschen, kopieren und finden
- Kenntnis von *Emacs* und *nano*
- Den Standardeditor festlegen

Liste wichtiger Kommandos und Optionen:

- `vi`
- `/`, `?`
- `h`, `j`, `k`, `l`
- `i`, `o`, `a`
- `d`, `p`, `y`, `dd`, `yy`
- `ZZ`, `:w!`, `:q!`
- `EDITOR`

Allgemeines

vi ist ein interaktiver Editor für Dateien, die im Textformat vorliegen. Er ist ein Nachfahre des Editors `ed`, weshalb er auch etliche Kommandos genauso unterstützt wie

sein Vorgänger. Es gibt inzwischen mehrere Derivate von *vi*, wie `vim`, `elvis`, `vile` usw. Es ist heutzutage sogar wahrscheinlich, dass auf Ihrem Linux-Computer `vim` als Editor gestartet wird, auch wenn Sie das Kommando `vi` eingeben. Der Befehl `vi` folgt dann einem Softlink, der auf `vim` zeigt. Wenn Sie `vi` bedienen können, werden Sie kaum in die Verlegenheit kommen, keinen Editor vorzufinden, den Sie bedienen können, wenn Sie sich mit einer Ihnen noch unbekannten Linux- oder Unix-Version befassen. Prüfungsfragen zum *vi* sind normalerweise nicht in großer Zahl zu erwarten. Leider sind diese Fragen dafür aber ziemlich schwierig, weil viele Optionen und Kommandos abgefragt werden, die Sie nicht unbedingt jeden Tag brauchen. Unterschätzen Sie also nicht die Relevanz der in den folgenden Abschnitten behandelten Themen.

Bedienungsgrundlagen

Start

Sie können den Editor einfach durch Ausführen des Kommandos `vi` starten oder gleichzeitig ein bzw. mehrere Dokumente laden. Mehrere Dokumente werden durch Leerzeichen voneinander getrennt angegeben:

```
archangel:/textfiles # vi text1 text2
```

Wenn mehrere Dokumente geladen wurden, können Sie mit `:n` (vorwärts) oder `:N` (rückwärts) zwischen den Dokumenten wechseln.

Modi

Wenn Sie den *vi* gerade gestartet haben, befindet sich der Editor im *Kommandomodus*. In diesem Modus können Sie den Text nicht direkt bearbeiten. Sie können sich lediglich im Text fortbewegen, Zwischenablagenoperationen durchführen, Löschungen vornehmen, nach Schlüsselwörtern suchen, die Datei speichern usw. Im *Einfügemodus* schreiben Sie Text wie mit jedem anderen Editor auch. Sie gelangen vom *Kommandomodus* in den *Einfügemodus*, indem Sie entweder die Taste `i` oder die Taste `a` betätigen.

- `i` wechselt vor der aktuellen Cursorposition in den Einfügemodus.
- `I` wechselt am Anfang der aktuellen Zeile in den Einfügemodus.
- `a` wechselt hinter der aktuellen Cursorposition in den Einfügemodus.
- `A` wechselt am Ende der aktuellen Zeile in den Einfügemodus.

Wie Sie sehen, gibt es bei der Auswahl der Kürzel einen logischen Zusammenhang, so dass Sie sich die Kommandos leicht merken können. Um wieder in den Kommandomodus zurückzugelangen, betätigen Sie einfach die Esc-Taste. Bei den meisten Versionen und Derivaten des *vi* wird in einer der unteren Ecken `--insert--` angezeigt,

wenn Sie sich im Einfügemodus befinden. Der Kommandomodus wird üblicherweise nicht gesondert angezeigt.

Beenden und speichern

Sie können den *vi* nur beenden oder ein Dokument speichern, wenn sich der Editor im Kommandomodus befindet. Folgende Kommandos sollten Sie in diesem Zusammenhang kennen:

- `:w dateiname` speichert die Datei unter *dateiname*. Sollte die Datei bereits existieren, ist die Angabe des Dateinamens optional.
- `:w!` speichert die Datei nach Möglichkeit auch dann, wenn sie schreibgeschützt ist.
- `:wq` speichert die aktuelle Datei und beendet den Editor.
- `:x` speichert die aktuelle Datei und beendet den Editor.
- `ZZ` speichert die aktuelle Datei und beendet den Editor. Der fehlende Doppelpunkt ist kein Druckfehler.
- `:q` beendet den Editor. Bearbeitete Dateien müssen vorher gespeichert worden sein.
- `:q!` beendet den *vi* auch dann, wenn Dateien noch nicht gespeichert wurden. Die Änderungen an diesen Dateien gehen verloren.

Sie sollten diese Kommandos auch dann schon kennen, wenn Sie den *vi* nur ausprobieren. Es ist nämlich nicht – wie in den meisten anderen Programmen – möglich, moderne Versionen des *vi* mit Strg + C zu beenden.

Navigation in einem Dokument

Normalerweise werden Sie mit den Cursortasten in einem Textdokument navigieren. Das ist auch sinnvoll, weil Sie ansonsten immer zuerst in den Kommandomodus wechseln müssten, um sich im Text fortzubewegen. Die Cursortasten funktionieren nämlich auch im Einfügemodus. Es gibt allerdings mindestens eine Situation, in der Sie wissen müssen, wie innerhalb des *vi* ohne Cursortasten navigiert wird, nämlich dann, wenn Sie eine Terminalsoftware verwenden, die den Editor-Block der Tastatur nicht korrekt übersetzt. Um diese Kommandos ausführen zu können, muss der Editor natürlich im Kommandomodus verwendet werden. Die wichtigsten *Bewegungskommandos* sind:

- `h` bewegt den Cursor um ein Zeichen nach links.
- `j` bewegt den Cursor um ein Zeichen nach unten.
- `k` bewegt den Cursor um ein Zeichen nach oben.
- `l` bewegt den Cursor um ein Zeichen nach rechts.
- `G` bewegt den Cursor zum Ende der Datei.

- `gg` bewegt den Cursor zum Anfang der Datei.
- `H` bewegt den Cursor zum oberen Bildschirmrand.
- `L` bewegt den Cursor zum unteren Bildschirmrand.

Sie können diesen Buchstaben auch Zahlen voranstellen, um mehrere Bewegungen auf einmal auszuführen. So wird `20j` den Cursor um 20 Zeilen nach unten verschieben. Es gibt zudem die Möglichkeit, Bewegungen zu kombinieren. Wenn Sie z. B. um drei Zeilen nach oben und um vier Zeichen nach rechts navigieren wollen, verwenden Sie `3k4l`. Der *vi* wird die Kommandos `3k` und `4l` dann allerdings getrennt ausführen.

Es gibt noch viele andere Zeichen, mit denen die Cursorbewegung (wortweise, satzweise usw.) gesteuert werden kann. Diese sind aber für die LPI-Prüfung nicht von Bedeutung.

Einfügen, Löschen, Kopieren und Auffinden von Text

Im Kommandomodus stehen Ihnen viele Möglichkeiten zur Bearbeitung eines Textes zur Verfügung. Die folgenden Kommandos sollten Sie unbedingt kennen:

- `yy` (yank) kopiert den Inhalt der aktuellen Zeile in die Zwischenablage.
- `p` (paste) fügt den Inhalt der Zwischenablage unterhalb der Zeile, in der sich der Cursor befindet, ein.
- `P` (Paste) fügt den Inhalt der Zwischenablage oberhalb der Zeile, in der sich der Cursor befindet, ein.
- `dd` (delete) löscht die komplette Zeile, in der sich der Cursor befindet.
- `D` (Delete) löscht den Text hinter dem Cursor bis zum Ende der Zeile.
- `cc` (cut) schneidet die komplette Zeile aus, in der sich der Cursor befindet. Die ausgeschnittene Zeile wird in die Zwischenablage kopiert. Anschließend wechselt *vi* automatisch in den Einfügemodus.
- `C` (Cut) schneidet den Text hinter dem Cursor bis zum Ende der Zeile aus. Der ausgeschnittene Text wird in die Zwischenablage kopiert. Anschließend wechselt *vi* automatisch in den Einfügemodus.
- `o` generiert eine neue Zeile unterhalb der Zeile, in der sich der Cursor aktuell befindet. Anschließend wechselt *vi* automatisch in den Einfügemodus.
- `O` generiert eine neue Zeile oberhalb der Zeile, in der sich der Cursor aktuell befindet. Anschließend wechselt *vi* automatisch in den Einfügemodus.
- `/regexp` sucht nach der Zeichenkette *regexp*.
- `/` sucht nach dem nächsten Vorkommen der Zeichenkette *regexp*.
- `?` sucht rückwärts nach dem vorherigen Vorkommen der Zeichenkette *regexp*.

- `:!kommando` führt das angegebene Shell-Kommando aus. Der Editor bleibt hierbei weiterhin geöffnet.

Soll ein Kommando mehrfach ausgeführt werden, wird diesem Kommando einfach ein numerischer Wert vorangestellt, der die Anzahl der Ausführungen des Befehls festlegt. Beispiel:

- `17dd` löscht 17 Zeilen (die aktuelle Zeile und die 16 folgenden).
- `54yy` kopiert 54 Zeilen in die Zwischenablage.
- `12cc` schneidet die aktuelle und die elf folgenden Zeilen aus.

Befehlszeilenoptionen für vi

Sie können bereits beim Starten des Editors einige anfängliche Bedingungen festlegen. Bei der Prüfung wird es sich bei diesen anfänglichen Bedingungen wahrscheinlich um die Cursorposition handeln. Wenn Sie die Datei *text1* so öffnen wollen, dass der Cursor gleich in Zeile 65 positioniert wird, geben Sie folgendes Kommando ein:

```
archangel:/textfiles # vi text1 +65
```

Eine andere Variante ist die Positionierung des Cursors mit einem Suchmuster. Wenn Sie die Datei *text1* so öffnen wollen, dass der Cursor sofort in der ersten Zeile erscheint, in der beispielsweise das Wort *Pinguin* gefunden wird, geben Sie diesen Befehl ein:

```
archangel:/textfiles # vi text1 +/Pinguin
```

Diese Befehlszeilenoptionen wurden vom Editor `ex` übernommen.

Prüfungstipp

Die in den letzten Abschnitten besprochenen Kommandos für den *vi* wurden sehr sorgfältig und prüfungsnah ausgesucht. Sie sollten zumindest diese Kommandos kennen, wenn Sie zur Prüfung gehen.

Emacs und nano

Zwei häufig anzutreffende Texteditoren sind *Emacs* und *Nano*. *Nano* ist in aller Regel in einer Standardinstallation enthalten und wird heutzutage oft als Standardeditor verwendet. Die Bedienung des Editors *nano* erschließt sich praktisch von selbst. Bei *Emacs* sieht das etwas anders aus. Hierbei handelt es sich um einen erweiterbaren Editor, für den es viele Zusatzmodule gibt. Das geht so weit, dass *Emacs* sogar über einen integrierten Browser, einen IMAP-tauglichen Email-Client und vieles mehr enthält. Die Syntax verschiedener Programmiersprachen wird farblich dargestellt.

Sie müssen in der Prüfung nicht mit Detailfragen zu diesen beiden Programmen rechnen, aber ich empfehle Ihnen trotzdem, sich zumindest mit ihnen auseinanderzusetzen.

Standardeditor

Der Standardeditor ist, falls konfiguriert, in der Variablen *EDITOR* abgelegt. Es ist aber von Distribution zu Distribution unterschiedlich, wie der Aufruf über das Kommando `editor` tatsächlich funktioniert. Unter Debian können Sie den Standardeditor für alle User festlegen, indem Sie dieses Kommando verwenden:

```
root@archangel:~# update-alternatives --config editor
Es gibt 5 Auswahlmöglichkeiten für die Alternative editor (welche /usr/bin/
editor bereitstellen).

  Auswahl      Pfad                Priorität  Status
------------------------------------------------------------
* 0            /bin/nano            40        automatischer Modus
  1            /bin/ed             -100       manueller Modus
  2            /bin/nano            40        manueller Modus
  3            /usr/bin/mcedit      25        manueller Modus
  4            /usr/bin/vim.basic   30        manueller Modus
  5            /usr/bin/vim.tiny    10        manueller Modus

Drücken Sie die Eingabetaste, um die aktuelle Wahl[*] beizubehalten,
oder geben Sie die Auswahlnummer ein:
```

Da ein normaler User dieses Kommando nicht verwenden darf, muss er, um den Standardeditor zu ändern, die Datei *.bashrc* bearbeiten. Hier könnten entsprechend folgende Zeilen hinzugefügt werden:

```
EDITOR=/usr/bin/vim
export EDITOR
alias editor=$EDITOR
```

Die erste Zeile setzt die Variable *EDITOR* auf */usr/bin/vim*. In der zweiten Zeile wird diese Variable exportiert und die dritte Zeile sorgt mit einem Alias dafür, dass bei Eingabe des Kommandos `editor` das in der Variablen *EDITOR* hinterlegte Programm ausgeführt wird.

Übungsfragen zu LPI 101

Die folgenden Fragen sollen Ihnen helfen, sich an die Art der Fragestellung in der wirklichen Prüfung zu gewöhnen. Es hat keinen Zweck, die Fragen einfach auswendig zu lernen, denn es sind keine echten Prüfungsfragen. Sie sollten versuchen, die Antworten zu jeder einzelnen Frage zu verstehen. Deshalb werden sowohl die richtigen als auch die falschen Antworten im Lösungsteil des Buches detailliert besprochen. Das Üben mit diesen Fragen soll Sie auch an die Herangehensweise an Ihnen eventuell unbekannte Themen näher bringen. Ein unbekanntes Kommando in einer Frage ist nämlich noch längst kein Grund, eine Frage einfach nicht zu beantworten. Oft führt ein wenig Logik oder das Ausschlussverfahren dennoch zum Ziel.

Fragen

Frage 1:

Sie versuchen, mithilfe des Kommandos `netstat -an` herauszubekommen, welche Ports ein Computer abhört. Leider bekommen Sie als Ergebnis eine lange unübersichtliche Auflistung. Welcher der folgenden Befehle wird die gewünschten Informationen übersichtlich darstellen?

- ☐ A: `netstat -an && grep LISTEN`
- ☐ B: `netstat -an | grep LISTEN`
- ☐ C: `netstat -an = grep LISTEN`
- ☐ D: `grep LISTEN |netstat -an`
- ☐ E: `grep netstat -l`

Frage 2:

Sie müssen herausfinden, welche Verzeichnisse und Dateien eines Webservers besonders oft besucht werden. Zu diesem Zweck wollen Sie zunächst die Logdatei des Webservers so filtern, dass Sie eine entsprechende Dateiliste erhalten, die nur noch die Verzeichnis- und Dateinamen enthält. Welches der folgenden Kommandos löst diese Aufgabe?

- ☐ A: `cut -f 7 /var/log/apache2/access.log > dateiliste`
- ☐ B: `cut -d " " /var/log/apache2/access.log > dateiliste`
- ☐ C: `cut -d " " -f 7 /var/log/apache2/access.log > dateiliste`
- ☐ D: `tail /var/log/apache2/access.log`
- ☐ E: `cat /var/log/apache2/access.log`

Frage 3:

Sie müssen das Release eines laufenden Kernels ermitteln. Mit welchem Kommando erhalten Sie ausschließlich diese Information?

- ☐ A: `uname -a`
- ☐ B: `uname -r`
- ☐ C: `ver -l`
- ☐ D: `version -k`

Frage 4:

Nach der Integration eines neuen Moduls in den laufenden Kernel wollen Sie überprüfen, ob der Kernel das Modul wie erwartet geladen hat. Welche der folgenden Kommandos können Sie dafür verwenden? (Wählen Sie zwei Antworten.)

- ☐ A: `cat /proc/modules`
- ☐ B: `insmod`
- ☐ C: `rmmod`
- ☐ D: `lsmod`

Frage 5:

Sie wollen dem Autor des Kernel-Moduls *nfs* eine E-Mail senden. Welches Kommando ist geeignet, um Informationen über den zuständigen Programmierer zu erlangen?

- ☐ A: `modinfo -a nfs`
- ☐ B: `insmod nfs`
- ☐ C: `lsmod`
- ☐ D: `modprobe`

Frage 6:

Welches Programm ist geeignet, Module des laufenden Kernels zu entladen? (Geben Sie nur das Programm ohne Optionen an.)

_____

Frage 7:

Sie beabsichtigen, ein Modul in den laufenden Kernel zu installieren. Leider wissen Sie nicht, ob das zu installierende Modul von anderen Modulen abhängt. Welches Kommando löst bei der Installation von Modulen die Abhängigkeiten automatisch auf?

- ☐ A: `insmod -auto`
- ☐ B: `autoinsmod`
- ☐ C: `depmod`
- ☐ D: `modprobe`

Frage 8:

Sie beabsichtigen, ein Modul aus dem laufenden Kernel zu entfernen. Mit welchen Programmen können Sie diese Aufgabe durchführen? (Wählen Sie zwei Antworten aus.)

- ☐ A: `rpm`
- ☐ B: `insmod`
- ☐ C: `modprobe`
- ☐ D: `rmmod`

Frage 9:

Welche der folgenden Programme sind auch bei etwas älteren Linux-Distributionen geeignet, *GUID*-Partitionstabellen zu erstellen bzw. zu modifizieren? (Wählen Sie zwei Antworten.)

- ☐ A: `fdisk`
- ☐ B: `gdisk`
- ☐ C: `parted`
- ☐ D: `mkfs`

Frage 120:

Sie planen die Partitionierung eines neuen Systems. Zur Vereinfachung der Datensicherung wollen Sie den Verzeichnissen, die besonders oft gesichert werden müssen, eine eigene Partition zuteilen. Welchen beiden der folgenden Verzeichnisse sollte unter diesen Gesichtspunkten möglichst eine eigene Partition zugewiesen werden?

- ☐ A: */etc*
- ☐ B: */home*
- ☐ C: */var*
- ☐ D: */boot*
- ☐ E: */proc*

Antworten und Erklärungen zu den Prüfungsfragen

Hier finden Sie die Erläuterungen zu allen Fragen des ersten Teils. Sie sollten unbedingt auch die Kommentare zu den falschen Antworten lesen. Einige Fakten werden hier nicht zufällig mehrfach erwähnt, sondern weil wesentliche Prüfungsinhalte auf diese Weise besser in Ihrem Gedächtnis haften bleiben.

Frage 1:

B: `netstat -an | grep LISTEN` liefert das gewünschte Ergebnis. Die sogenannte *Pipe* (der senkrechte Strich) sorgt dafür, dass die Ausgabe des Kommandos `netstat` an `grep` übergeben wird. Das Programm `grep` sucht dann im Ergebnis nach der Zeichenkette `LISTEN`.

zu A: Hier würde zunächst `netstat -an` ausgeführt werden und im Anschluss würde `grep` in *stdin* nach der Zeichenkette `LISTEN` suchen.

Die anderen Antworten sind frei erfunden und ergeben noch weniger Sinn.

Frage 2:

C: `cut -d " " -f 7 /var/log/apache2/access.log > dateiliste`

Mit diesem Kommando wird die siebte Spalte aus der Logdatei ausgeschnitten und in die Datei *dateiliste* geschrieben. Der Delimiter ist in dieser Logdatei das Leerzeichen zwischen den Double Quotes.

zu A: `cut -f 7 /var/log/apache2/access.log > dateiliste` – ohne die Angabe eines Delimiters wird `cut` in diesem Fall die ganze Datei ausgeben.

zu B: `cut -d " " /var/log/apache2/access.log > dateiliste` – wenn kein Feld angegeben wird, gibt `cut` lediglich eine Fehlermeldung aus.

zu D: `tail /var/log/apache2/access.log` gibt die letzten 10 Zeilen der Datei aus.

zu E: `cat /var/log/apache2/access.log` gibt die komplette Datei ungefiltert aus.

Frage 3:

B: `uname -r` zeigt genau die gewünschte Information an.

zu A: `uname -a` würde zusätzliche, nicht benötigte Informationen ausgeben.

Die Kommandos der Antworten C und D gibt es nicht. Sie sind frei erfunden.

Frage 4:

A: `cat /proc/modules` gibt Informationen über geladene Module direkt aus dem Proc-device aus.

D: `lsmod` ist das Kommando zum Auflisten der geladenen Kernel-Module.

zu B: `insmod` fügt dem laufenden Kernel Module hinzu.

zu C: `rmmod` entfernt Module aus dem laufenden Kernel.

Frage 5:

A: `modinfo -a nfs` zeigt den Autor des Moduls inkl. der E-Mail-Adresse (falls hinterlegt) an.

zu B: `insmod nfs` fügt das Modul in den laufenden Kernel ein.

zu C: `lsmod` listet die Module des laufenden Kernels auf.

zu D: `modprobe` ist ein Programm zum Laden bzw. Entladen von Modulen.

Frage 6:

`rmmod` und `modprobe` sind beide geeignet, diese Aufgabe durchzuführen.

Frage 7:

D: `modprobe` ist geeignet, Module in den laufenden Kernel zu laden und hierbei gegebenenfalls Abhängigkeiten aufzulösen.

zu A: `insmod` ist der Vorläufer von `modprobe` und kann Abhängigkeiten nicht automatisch auflösen.

zu B: `autoinsmod` gibt es nicht.

zu C: `depmod` generiert die Datei *modules.dep*.

Frage 8:

C: `modprobe` und D: `rmmod` sind beide für diesen Vorgang geeignet.

zu A: `rpm` ist ein Programm zur Paketverwaltung, also praktisch völlig am Thema vorbei.

zu B: `insmod` fügt ein Kernel-Modul ein. Das ist aber das Gegenteil von dem, was gefordert wurde.

Frage 9:

B: `gdisk` wird verwendet, um eine *GUID*-Partitionstabelle zu erzeugen und darin Partitionen zu erstellen oder zu löschen.

C: `parted` kann verwendet werden, um Partitionen zu ändern. Es unterstützt auch *GUID*-Partitionstabellen.

zu A: `fdisk` ist primär auf Partitionen auf *MBR*-Basis ausgerichtet. Dieses Programm unterstützt erst bei relativ neuen Distributionen die Verwaltung von *GPT*-Tabellen.

zu D: `mkfs` ist nur für die Erstellung von Dateisystemen zuständig. Partitionen werden damit weder angelegt noch in anderer Weise manipuliert.

Frage 10:

`zypper` installiert bei *openSUSE* die Pakete. Die Antwort *Zypper* wäre ebenfalls akzeptabel, wenn man über das Produkt und nicht über das Kommando spricht.

Frage 11:

D: `blkid` zeigt eine Aufstellung der blockorientierten Geräte inklusive deren *UUID*s.

zu A: `fdisk` und zu B: `gdisk` können, wenn im interaktiven Modus die Option `i` verwendet wird, ebenfalls *UUID*s anzeigen, aber sie erstellen keine Liste.

zu C: `lsblk` kann, wenn Sie die Option `-o +UUID` verwenden, auch zum Ziel führen, aber es gibt diese Antwortmöglichkeit nicht.

Prüfungstipp

Manchmal kommt es auf die genaue Formulierung der Frage an. Lesen Sie immer sorgfältig.

Frage 12:

C: */usr/src/linux* ist normalerweise ein Softlink auf die Quellen des laufenden Kernels. Die Quellen selbst befinden sich im Normalfall ebenfalls in einem eigenen Unterverzeichnis von */usr/src*.

zu A: */boot* enthält hauptsächlich den statischen Teil des Kernels und den Bootloader *GRUB*.

Auf einen Blick

LPI 101

LPI 102

Inhalt

104 Geräte, Linux-Dateisysteme, Filesystem Hierarchy Standard 167

LPI 102