

SelfLinux-0.13.0



Elementare Informationen



Autor: Frank Börner (*f.boerner@ngi.de*)
Autor: Ferdinand Hahmann (*FerdinandHahmann@gmx.net*)
Formatierung: Florian Frank (*florian.frank@pingos.org*)
Formatierung: Matthias Hagedorn (*matthias.hagedorn@selflinux.org*)
Lizenz: GFDL

Inhaltsverzeichnis

1 Benutzerinformationen

- 1.1 Einführung
- 1.2 id
- 1.3 logname
- 1.4 who
- 1.5 w
- 1.6 finger

2 Informationen über die Speicherbelegung

- 2.1 free
- 2.2 df
- 2.3 du

3 Weitere Kommandos

- 3.1 dmesg
- 3.2 date
- 3.3 dd
- 3.4 which

4 Bearbeitung von Programmausgaben

1 Benutzerinformationen

1.1 Einführung

Die folgenden Beispiele sind unter **X** größtenteils nicht funktionsfähig, weil die Kommandos nicht herausfinden können, welcher Benutzer angemeldet ist. Darum ist die Verwendung einer Textkonsole empfehlenswert.

Wenn Sie gerade unter **X** arbeiten, können Sie mit der Tastenkombination `Strg + Alt + F1` bis `Strg + Alt + F6` auf eine Textkonsole wechseln. Hier werden Sie sich nun vermutlich erst anmelden müssen. Geben Sie wie üblich Ihren Benutzernamen und Ihr Passwort ein. Um wieder zu **X** zu wechseln genügt `Alt + F7`.

1.2 id

`id` gibt Informationen über die Daten, mit denen die Benutzer eines Linux-Systems verwaltet werden, aus. Jeder Benutzer hat eine Benutzernummer, die sogenannte **UID**. Zudem ist er Mitglied in einer oder mehreren Gruppen von Benutzern, was wichtig für die Regelung von Zugriffsrechten und Rechten zur Nutzung von Systemressourcen ist.

```
user@linux / $ id
uid=500(penguin) gid=100(users) groups=100(users),14(uucp)
```

Unser Pinguin ist der Benutzer mit der Nummer 500, und seine wichtigste Gruppe ist die Gruppe 100 mit dem Namen **users**. Auf fast jedem Linux-System gibt es diese Gruppe, und normalerweise sind alle Benutzer Mitglied in ihr.

Er gehört auch noch einer anderen Gruppe an, der Gruppe **uucp**, die traditionell das Recht gibt, über Modemverbindungen Daten mit der Außenwelt auszutauschen.

`id` kann auch Informationen über andere Benutzer auf dem System liefern:

```
user@linux / $ id root
uid=0(root) gid=0(root)
groups=0(root),1(bin),14(uucp),15(shadow),16(dialout),17(audio),65534
(nogroup)
```

Dies ist der Systemverwalter **root**. Er hat seine eigene Gruppe.

1.3 logname

Mit diesem Kommando wird der Benutzername des Benutzers abgefragt, der es aufruft. `logname` lässt sich auch von Kommandos, die es einem Benutzer ermöglichen, vorübergehend einen anderen Namen anzunehmen, nicht irritieren:

```
user@linux / $ logname
```

```
penguin
user@linux / $ su
Kennwort:
root@linux / # logname
penguin
root@linux / # exit
user@linux / $ logname
penguin
```

1.4 who

Linux ermöglicht es Benutzern, sich über ein Netzwerk (das kann auch ein Modem sein) am System anzumelden und zu arbeiten. Daher können mehrere Benutzer zur gleichen Zeit angemeldet sein. Das Kommando `who` gibt einen Überblick über die angemeldeten Benutzer:

```
user@linux / $ who
penguin          tty1                Jan 18 22:27
eisbaer          tty3                Jan 20 07:45
```

Dies bedeutet, dass der Pinguin auf der ersten Textkonsole angemeldet ist, und der Eisbär auf der dritten. In der dritten Spalte steht der Zeitpunkt der Anmeldung.

```
user@linux / $ who am i
penguin  tty1                Jan 18 22:27
```

`who am i` (**Wer bin ich**) zeigt nur den Benutzer an, der das Kommando aufgerufen hat. Diese Angaben lassen sich in Shell-Skripten leicht auswerten.

Zwei interessante Optionen sind `-H` und `--login`.

- * `-H` (headline) gibt zusätzlich noch eine Kopfzeile mit aus, wie Sie es von den meisten anderen Programmen gewöhnt sind.
- * `--login` gibt nicht die Benutzer, die gerade angemeldet sind aus, sondern zeigt an, wo gerade ein Login-Prozess läuft.

1.5 w

`w` ist wesentlich gesprächiger als `who`. Es gibt alle Informationen, die auch `who` liefert, und zusätzlich noch statistische Daten über Rechenzeitverbrauch und gerade aktive Programme:

```
user@linux / $ w
```

```
9:49pm up 14 days, 4:29, 2 users, load average: 1.08, 1.02, 1.01
USER      TTY      FROM          LOGIN@      IDLE        JCPU        PCPU        WHAT
penguin   tty1     -             Tue10pm    2days     0.42s     0.31s     -bash
eisbaer   pts/3    sp.antarktis.net 9:49pm    0.00s     0.32s     0.06s     w
```

Diese Zeilen enthalten:

- * Die aktuelle Uhrzeit. (9:49 pm = 21:49 h)
- * Die Uptime: Das System läuft seit zwei Wochen, vier Stunden und 29 Minuten. (up 14 days, 4:29)
- * Die Anzahl der angemeldeten User: 2 (2 users)
- * Die Systemauslastung in der letzten Minute, den letzten fünf Minuten und den letzten 15 Minuten. (load average: 1.08, 1.02, 1.01)
- * Zusätzlich zu den von `who` bekannten Daten in den Spalten USER, TTY und LOGIN@ noch:
- * FROM: Den Namen des Computers, von dem aus sich der Benutzer angemeldet hat. Der Pinguin sitzt an dem Computer selbst, der Eisbär könnte irgendwo sein, denn er ist über ein Netzwerk angemeldet.
- * IDLE: Die Zeit, seit der jeweilige Benutzer zuletzt etwas getan hat: Pinguin ist schon seit zwei Tagen untätig, er wird wahrscheinlich vergessen haben, sich abzumelden.
- * JCPU: Die Rechenzeit, die alle zum gegenwärtigen Zeitpunkt laufenden Programme bisher belegt haben. Hier lässt sich erkennen, ob ein Benutzer ein Programm gestartet hat, das viel Rechenzeit verbraucht. Der Systemverwalter kann dann den Benutzer auffordern, das Programm zu beenden, er kann dem Programm weniger Rechenzeit zuteilen, oder er kann es eigenmächtig beenden.
- * WHAT: Das Programm, das der Benutzer zuletzt gestartet hat, wenn es nicht im Hintergrund abläuft.
- * PCPU: Die bisher von dem unter WHAT angegebenen Programm verbrauchte Rechenzeit.

1.6 finger

Um mehr über einen Benutzer zu erfahren, gibt es das Tool `finger`. Wie viel Informationen `finger` ausgibt, hängt davon ab, wie viel der Systemverwalter und der Benutzer zulässt. `finger` sucht unter anderem nach Informationen in den Dateien `~/.plan` und `~/.project`. Damit `finger` die Dateien `.plan` und `.project` ausgeben kann, muss `finger` darauf zugreifen können. Da diese Dateien im `home`-Verzeichnis liegen, ist dieser Umstand nicht immer gegeben.

Ohne Angabe eines Benutzernamens wird eine Kurzinformation über alle angemeldeten Benutzer ausgegeben.

```
user@linux / $ finger
Login  Name      Tty      Idle      Login Time      Office  Office Phone
penguin  *tty1    1:45 Jul 26 9:56
eisbaer  *tty2    7 Jul 26 11:18
```

- * Login: Hiermit ist der Login-Name gemeint.
- * Name: Das ist der reale Name des Users. Bei beiden wurde kein Name angegeben.
- * Tty: Hier ist das Terminal angegeben, von wo aus sich der Benutzer angemeldet hat.
- * Idle: Gibt wie auch bei `w` die Zeitspanne an seit dem der Benutzer inaktiv ist.
- * Login Time: Das ist die Angabe, wann er sich angemeldet hat.
- * Office: Gibt Informationen zum Büro, z. B. Adresse aus.
- * Office Phone: Hier steht, wenn vorhanden, die Telefonnummer des Büros, in welchem der Benutzer arbeitet.

Gibt man einen Benutzernamen mit an, werden genauere Informationen über diesen einen Benutzer ausgegeben.

```
user@linux / $ finger pinguin
Login: pinguin                Name: (null)
Directory: /home/pinguin      Shell: /bin/bash
On since Sat Jul 26 11:08 (CEST) on tty2      8 minutes 2 seconds idle
      (messages off)
No mail.
No Plan.
```

- * **Login** und **Name** wurden schon weiter oben erklärt.
- * **Directory** gibt das **home**-Verzeichnis des Benutzers an. In diesem Fall ist es wie für Benutzer üblich `/home/pinguin`.
- * **Shell** gibt die **Standardshell** des Benutzers an. Pinguin benutzt `/bin/bash`.
- * Danach erfolgt die Angabe, seit wann und wo der Benutzer angemeldet ist, gefolgt von der Idle-Zeit.
- * **messages off** sagt aus, dass der Benutzer keine Nachrichten empfangen kann.
- * **No mail** bedeutet dass der Benutzer keine Mails in seinem Mailordner hat. Wenn ungelesene Mails vorhanden sind, wird zum dem noch angegeben, seit wann die Mails nicht gelesen wurden.
- * **No Plan** sagt aus, dass die Datei `.plan` nicht vorhanden ist oder `finger` darauf keinen Zugriff hat.

2 Informationen über die Speicherbelegung

2.1 free

Von Zeit zu Zeit ist es nützlich, die Speicherbelegung in Augenschein zu nehmen, beispielsweise weil ein Programm nicht genug Speicher bekommt, oder wenn der Systemverwalter sich für die Auslastung des Auslagerungsbereiches interessiert.

```
user@linux / $ free
```

	total	used	free	shared	buffers	cached
Mem:	128284(1)	117228(2)	11056(3)	25644(4)	5736(5)	64304(6)
-/+ buffers/cache:		47188(7)	81096(8)			
Swap:	128480(9)	13376(10)	115104(11)			

(Die Zahl in Klammern dient lediglich der Erklärung und ist keine Ausgabe von free.)

Diese vielen Zahlen bedeuten:

1. Die gesamte für das System verfügbare Speichermenge. Hier ist der größte Teil von 128 MB verfügbar, weil der vom Kernel belegte Speicherplatz nicht mitgerechnet wird.
2. Die Menge des belegten Speichers.
3. Die Menge des freien Speichers.
4. Die Größe von zwischen Prozessen geteilten Speicherbereichen.
5. Der für verschiedene Arten von Zwischenspeichern (Caches) verwendete Speicher. Der Unterschied ist nicht von Bedeutung, wenn man nicht gerade am Kernel programmiert.
6. siehe 5.
7. Der belegte Speicher nach Abzug aller Zwischenspeicherarten. Diese Speichermenge ist tatsächlich von Programmen belegt.
8. Der freie Speicher nach Abzug aller Zwischenspeicherarten. Dieser Speicher ist noch für Programme verfügbar.
9. Die Gesamtgröße der vorhandenen Auslagerungsbereiche. Hier sind es knapp 128 MB.
10. Der genutzte Teil des Auslagerungsbereiches. Untätige Prozesse werden in den Auslagerungsbereich verschoben, wenn der von ihnen belegte Arbeitsspeicher besser für andere Zwecke verwendet werden kann.
11. Der freie Teil des Auslagerungsbereiches.

Mit den Optionen `-b`, `-k` und `-m` wird der Speicher in Byte, KByte, bzw. in MByte ausgegeben. Als Standard-Einstellung wird KByte verwendet.

Die Option `-t` (total) gibt zusätzlich noch die Summe der Gesamtgrößen aus.

2.2 df

Eines der häufigsten Probleme beim Betrieb eines Linux-Systems, und noch dazu eines, das sich oft nicht rechtzeitig zu erkennen gibt, ist eine vollgelaufene Festplatte. Heutige Linux-Distributionen enthalten derartige Mengen an Software, dass es überhaupt kein Problem ist, auch eine Fünf- oder mehr Gigabyte-Partition in

kürzester Zeit zu füllen.

`df` zeigt den Belegungszustand jedes eingehängten Dateisystems an.

```
user@linux / $ df
Filesystem      1k-blocks      Used Available Use% Mounted on
/dev/sda2        208820        76912   131908   37% /
/dev/sda5        763012       104468   658544   14% /var
/dev/sda3       5245048      3128612  2116436   60% /usr
/dev/sda1         7988         2404     5184    32% /boot
ser1:/home/pinguin 2097286    1305429   686633   66% /home/pinguin
```

In der ersten Spalte (Filesystem) steht die Bezeichnung des Dateisystems, meistens eine Festplattenpartition. Dahinter steht die Gesamtgröße in Kilobyte (1k-blocks), der belegte Speicher (Used) und der noch vorhandene freie Platz (Available). Außerdem noch der Anteil des belegten Speichers an der Gesamtgröße (Use%) und das Verzeichnis, in das das Dateisystem eingehängt ist (Mounted on).

In der letzten Zeile ist ein Dateisystem zu sehen, auf das über das Netzwerk zugegriffen wird.

Mit der Option `-a` werden auch Dateisysteme angezeigt, die eine Kapazität von 0 Byte haben. Damit sind Dateisysteme gemeint, die im Prinzip keinen Speicherplatz belegen, sondern eine bestimmte Funktionalität zur Verfügung stellen. Ein Beispiel ist das `devfs`-Dateisystem, welches nur Dateien beinhaltet, die Geräte darstellen.

Wenn man noch zu jedem Dateisystem dem Typ erfahren will, so gibt es dafür noch die Option `-T`.

Die Option `-i` zeigt anstelle der Speicherbelegung die Belegung der Inodes an. Unter Linux benötigt jede Datei eine bestimmte Inode-Nummer. Es gibt allerdings immer eine maximale Anzahl an Inode-Nummern pro Dateisystem. Wenn diese Anzahl erreicht ist, kann keine weitere Datei mehr angelegt werden, egal wie viel Speicher noch frei ist.

Wie Sie im Beispiel sehen konnten, sind die Zahlen zum Teil ziemlich unhandlich und somit weniger aussagekräftig. Aus diesem Grund gibt es die Optionen `-h`, bzw. `-H` (human-readable). Diese haben die Aufgabe, die Zahlen für den Menschen besser lesbar darzustellen. Der Unterschied zwischen den beiden Optionen besteht darin, dass `-h` mit einer Potenz von 1024 und `-H` von 1000 rechnet. Zudem stehen noch die Optionen `-k` und `-m` zur Verfügung. Diese geben den Plattenplatz in Kilobyte, bzw. in Megabyte aus. Mit `--block-size=n` erfolgt die Ausgabe in n-Byte-Blöcken.

2.3 du

Mit Hilfe von `du` (disk usage) wird der Speicherplatzverbrauch für ein Verzeichnis und dessen Unterverzeichnisse angezeigt. Als Standard-Einstellung wird das aktuelle Arbeitsverzeichnis verwendet.

```
user@linux / $ du
...
120      ./kde/share/apps/kMail
...
```

Beim Autor war die Ausgabe von `du` wesentlich länger. Diese eine Zeile soll exemplarisch betrachtet werden.

Das Beispiel sagt aus, dass das Verzeichnis `./kde/share/apps/kMail` insgesamt 120 Kilobyte belegt (als Standard-Einstellung erfolgt die Ausgabe in Kilobytes). Um die Ausgabe besser lesbar darzustellen, gibt es wie bei `df` die Optionen `-h` und `-H`.

Ebenfalls stehen wie bei `df` die Optionen `-k` und `-m` für eine Ausgabe in Kilo- bzw. Megabyte zur Verfügung. Hinzu kommt noch die Option `-b` für Byte.

Um nur die Gesamtsumme, die ein bestimmtes Verzeichnis belegt, zu erfahren, verwendet man die Option `-s`.

```
user@linux / $ du -s
40248 .
```

Auf den ersten Blick mag diese Ausgabe ein wenig ungewohnt erscheinen, haben wir doch erwartet, dass alle Verzeichnisse im aktuellem Verzeichnis ausgegeben werden. Bei genauerer Betrachtung ist die Ausgabe aber durchaus logisch. Als Standard-Einstellung wird das aktuelle Arbeitsverzeichnis genommen und das ist nun mal `./`.

Will man alle Unterverzeichnisse aufgelistet haben, so muss der Befehl `du -s *` lauten.

Um zusätzlich noch den belegten Speicher für jede Datei auszugeben, gibt es die Optionen `-a`, bzw. `--all`.

Die Option `-c` gibt am Ende noch die Gesamtsumme aus.

3 Weitere Kommandos

3.1 dmesg

Der Kernel gibt im Laufe der Zeit eine Menge Informationen an den **klogd**, den **kernel log daemon** weiter. **dmesg** zeigt die aktuellsten Meldungen an. Hier sind z. B. die Bootmeldungen nachzulesen, aber auch das Einlegen einer neuen CD wird hier vermerkt. Weil in diesen Meldungen auch all die Hardwareinformationen enthalten sind, die beim Hochfahren des Systems anfallen, ist **dmesg** für die Fehlersuche oder die Konfiguration sehr nützlich.

3.2 date

Wer möchte nicht gerne wissen, welcher Tag heute ist (vor allem nach stundenlangem Programmieren, vorzugsweise nachts). Dazu gibt **date** das aktuelle Datum und die Uhrzeit aus:

```
user@linux / $ date
Fr Jan 21 22:57:58 CET 2000
```

Manchmal kann es aber notwendig sein, sich das Datum in einem individuellem Format ausgeben zu lassen. Wenn dies der Fall ist, sieht der Befehl folgendermaßen aus:

`date +'Format'`

Format ist eine Zeichenfolge, die angibt, wie das ausgegebene Datum aussehen soll. Dabei stehen dem Benutzer verschiedene Platzhalter, die durch aktuelle Werte ersetzt werden, zur Verfügung. Die wichtigsten sind unter anderem:

- * %d: Aktueller Tag des Monats von 01 bis 31
- * %H: Aktuelle Stunde von 00 bis 23
- * %m: Aktueller Monat von 01 bis 12
- * %M: Aktuelle Minute von 00 bis 59
- * %Y: Aktuelles Jahr von 1970 bis ...

Beispiel:

```
user@linux / $ date +%Y-%m-%d
2000-01-21
```

Von Zeit zu Zeit will man sich aber auch das Datum in einem bestimmten standardisiertem Format ausgeben lassen. Dazu stehen folgen Optionen zur Verfügung:

- * **-I**[**TIMESPEC**], **--iso-8601**[=**TIMESPEC**]
Gibt das Datum im ISO 8601 Format aus. Als **TIMESPEC** kann **date** (gibt nur das Datum aus), **hours** (gibt zudem noch die Stunde an), **minutes** (die Minuten werden noch zusätzlich angezeigt) oder **seconds** (die Sekunden werden auch noch angezeigt) sein. Als Default für **TIMESPEC** ist **date** eingestellt.
- * **-R**, **--rfc-822**

Gibt das Datum im RFC 822 Format aus.

* `-u`, `--utc`, `--universal`

Gibt das Datum im UTC Format aus.

Besitzt man die nötigen administrativen Rechte, kann man mit `date` auch Datum und Uhrzeit ändern. Dazu gibt es die Option `-s`. Wichtig dabei ist dennoch ein Format anzugeben. Das Format gibt dann an in welcher Form das Datum übergeben wird. Die genaue Syntax lautet wie folgt:

```
date +'Format' -s "Datum"
```

bzw.

```
date +'Format' --set="Datum"
```

Dabei gibt 'Datum' das neue Datum an. Selbstverständlich kann anstelle von +'Format' auch eine der Optionen `-I`, `-R` oder `-u` verwendet werden.

3.3 dd

Die primäre Aufgabe von `dd` ist es Daten zu kopieren und entsprechend zu konvertieren. Dabei kopiert `dd` die Daten nicht Dateiweise, sondern Blockweise. Das Haupteinsatzgebiet von `dd` ist es Kopien von ganzen Devices auf einem anderen (physikalischen) Device anzulegen. Somit eignet sich `dd` auch dafür, ein ISO-Abbild von einer CD auf der Festplatte zu sichern.

`dd` besitzt eine Reihe von verschiedenen Optionen, die verschiedene Möglichkeiten der Konvertierung darstellen. Die beiden wichtigsten Optionen sind allerdings `if=DATEI` und `of=DATEI`. `if=DATEI` gibt dabei die Datei an, von der gelesen werden soll. `of=DATEI` gibt die Datei an, in die geschrieben werden soll.

Ein Beispiel, um ein ISO-Abbild auf der Festplatte im aktuellen Verzeichnis anzulegen, lautet:

```
root@linux / # dd if=/dev/cdrom of=cdrom.iso
```

Wie schon angesprochen kopiert `dd` nicht Dateiweise sondern Blockweise. Aus diesem Grund ist es auch möglich Daten zu kopieren und dabei die Blockgröße zu ändern. Ebenso ist es auch möglich nur bestimmte Blöcke von einem Device zu kopieren.

Um die Blockgröße für die Ein- und Ausgabe festzulegen, stehen drei Optionen zur Verfügung:

- * `ibs=n`: Legt die Eingabeblockgröße auf n Bytes fest.
- * `obs=n`: Legt die Ausgabeblockgröße auf n Bytes fest.
- * `bs=n`: Legt die Ein- und Ausgabeblockgröße auf n Bytes fest. `bs` hat Vorrang von `ibs` und `obs`.
- * `cbs=n`: Legt die Datensatzlänge auf n Bytes fest.

Die voreingestellte Größe für die Ein- und Ausgabeblöcke ist 512 Byte.

Um nur bestimmte Blöcke zu kopieren, stehen folgende Optionen zur Verfügung:

- * `skip=n`: Überspringt n-Blöcke am Anfang der Eingabedatei.
- * `seek=n`: Überspringt n-Blöcke am Anfang der Ausgabedatei
- * `count=n`: Kopiert nur n Eingabeblöcke.

Als Blockgröße wird logischerweise, der durch `bs`, `ibs`, bzw. `obs` angegeben Wert genutzt.

Eine weitere wichtige Option ist `conv=Key`. Wenn diese Option angegeben ist, wird die Eingabedatei entsprechend durch das mit Key angegebene Schlüsselwort in die Ausgabedatei konvertiert. Als Key können auch mehrere Schlüsselwörter, durch Kommata getrennt, angegeben werden.

Unter anderem stehen dem Benutzer dabei folgenden Schlüsselwörter zur Verfügung:

* **block/unblock**

Es gibt Datensätze fester Länge und es gibt Datensätze mit variabler Länge, deren Ende durch einen Zeilenumbruch markiert ist. Das Schlüsselwort **block** füllt einen Datensatz, der kleiner als cbs-Bytes ist mit Leerzeilen auf, bis die entsprechende Datensatzlänge erreicht ist. Somit wandelt **block** Datensätze variabler Länge in Datensätze mit fester Länge um. Entsprechend entfernt **unblock** die nachfolgenden Leerzeilen und wandelt somit Datensätze mit fester Länge in Datensätze mit variabler Länge um.

* **lcase**

Sämtliche Großbuchstaben werden in Kleinbuchstaben umgewandelt

* **ucase**

Sämtliche Kleinbuchstaben werden in Großbuchstaben umgewandelt

* **noerror**

Die Verarbeitung wird auch nach einem Fehler fortgesetzt.

3.4 which

Wenn Sie auf der Shell einen Befehl eingeben, werden der Reihe nach alle Verzeichnisse in **\$PATH** nach diesem Befehl durchsucht. Nachdem der Befehl in einem Verzeichnis gefunden wurde, wird die Suche abgebrochen. Dabei kann allerdings das Problem auftreten, dass Sie ein anderes Kommando meinen, das sich in einem anderen Verzeichnis befindet. Um herauszufinden, wo sich nun das Programm befindet, das ausgeführt wird, gibt es das Werkzeug `which`.

`which` durchsucht alle Verzeichnisse, die in der Umgebungsvariablen **\$PATH** aufgelistet sind, nach einer ausführbaren Datei mit dem angegebenen Namen:

```
user@linux / $ which man
/usr/bin/man
```

Auf diese Weise finden Sie heraus, ob das seltsame Verhalten eines Kommandos dadurch verursacht wird, dass ein anderes Programm in ein Verzeichnis geraten ist, das sich im Pfad weiter vorne befindet:

```
user@linux / $ man ls
cat: ls: No such file or directory
user@linux / $ which man
/usr/local/bin/man
```

Hier ist ein Programm namens `man` in dem Verzeichnis `/usr/local/bin`, das im Suchpfad vor dem Verzeichnis `/usr/bin` steht, in dem sich das gewünschte Kommando `man` befindet. Durch den Aufruf:

```
user@linux / $ /usr/bin/man ls
```

können Sie jetzt das richtige Programm aufrufen.

`which` kann zudem nützlich sein, um festzustellen, ob ein bestimmtes Programm vorhanden ist ohne dieses Programm gleich ausführen zu müssen.

4 Bearbeitung von Programmausgaben

Manchmal können Ausgaben von Programmen recht umfangreich werden. Um die Ausgaben ein wenig übersichtlicher zu gestalten, stehen dem Benutzer dieselben Werkzeuge mit denselben Optionen und derselben Funktionalität zur Verfügung wie bei der Dateiverwaltung, also z. B. `less`, `grep` usw. Dabei muss man die Ausgabe von dem einem Programm per [Pipe](#) in ein anderes Programm weiterleiten, z. B.:

```
user@linux / $ ps -ax | less
```

Der Befehl `ps -ax` wird ganz normal ausgeführt, aber die Ausgabe dient in diesem Fall als Eingabe für `less`. Aus diesem Grund muss auch kein Dateiname für `less` angegeben werden, da die Ausgabe von `ps -ax` von `less` wie eine Datei behandelt wird. Das Resultat dieser Kombination ist, dass man durch die gesamte Ausgabe von `ps -ax` bewegen kann und man sich nicht nur mit den letzten Zeilen zu Frieden geben muss.

Ausführliche Informationen zu `ps` findet man unter [ps](#).