Die Welt der (Unix-)Shell

harald@ccbib.org

Dieser Text soll die wichtigsten Grundlagen zusammenfassen und trotzdem so kurz sein, dass er leicht vollständig gelesen werden kann. Auch ohne alle Details zu verstehen wird der Überblick helfen, im Anlassfall schneller die gesuchte Information zu finden.

Einzig die Befehle `man` und `apropos` sind essenziell. Der Rest ist hoffentlich als Merkzettel nützlich.

Der aktuellste Sourcecode für diesen Zettel ist unter:
`http://repo.or.cz/eineseite.git`
bzw. `git://repo.or.cz/eineseite.git`

Das Dateisystem

Unter Unixsystemen gibt es keine Laufwerke. Alle Dateien sind in einer abstrakten Hierarchie angeordnet, unabhängig davon auf welchem Gerät die Daten tatsächlich gespeichert sind. Das mag für Umsteiger ungewohnt wirken, hat aber den großen Vorteil, dass das Dateisystem auf allen Unixsystemen sehr ähnlich aufgebaut ist. Dadurch ist es leicht sich auf neuen Computern zurechtzufinden bzw. bei vielen Servern den Überblick nicht zu verlieren.

In einem Dateiort (Pfad) werden Verzeichnisnamen durch Slash („/“) getrennt. Das oberste Verzeichnis in der Hierarchie (sozusagen die Wurzel des Dateisystembaums – auch root-Verzeichnis genannt) wird nur mit einem „/“ bezeichnet. In den darunterliegenden Verzeichnissen befinden sich die Systemdateien. Im Folgenden werden die wichtigsten Verzeichnisse kurz angeführt:

`/etc` Hier befinden sich die Konfigurationsdateien des Systems. Das sind üblicherweise kleine Textdateien, die mit einem beliebigen Editor geöffnet werden können.

`/bin`, `/sbin` In diesen beiden Verzeichnissen befinden sich die wichtigsten Programme, ohne die das System nicht starten kann. Die Programme in /sbin sind nur für `root` vorgesehen.

`/dev` Auf unixartigen Systemen werden auch Geräte wie z. B. (Festplatten, Mäuse, USB-Sticks, ...) als Dateien repräsentiert – dafür ist dieses Verzeichnis vorgesehen. Grundsätzlich kann darauf auch ähnlich wie auf normale Dateien zugegriffen werden, sofern das verwendete Programm etwas mit den Rohdaten anfangen kann.

`/lib` In diesem Verzeichnis befindet sich ausführbarer Code, der von verschiedenen Programmen benötigt wird.

`/home` Der Ort für die Daten der Benutzer. Es gibt kein wirklich einheitliches Format, aber üblich ist, dass es für jeden Benutzer ein Unterverzeichnis (sein sogenanntes HOME-Verzeichnis) gibt, in dem alle seine (privaten) Daten liegen.

`/var/log` In `/var` speichern Programme ihre Betriebsdaten, um die sich der Benutzer normalerweise nicht kümmern muss. `/var/log` ist von besonderem Interesse: Hier liegen zahlreiche Logfiles. Das sind Dateien, in denen protokolliert wird, was das System gerade macht. Wenn irgendetwas nicht funktioniert, dann finden sich in den Logfiles oft wichtige Hinweise auf die Ursache.

`/proc` In diesem Verzeichnis sind keine normalen Dateien (vgl. mit `/dev`) sondern „virtuelle Dateien“, die das System beschreiben: Welche Treiber sind geladen, welche Hardware wurde erkannt, welche Programme laufen gerade, ...

`/usr` Der Großteil des Systems, also zum Beispiel alle Programme die nicht zum Booten des Rechners notwendig sind. Der Aufbau erinnert ein bisschen an das Wurzelverzeichnis selbst.

`/usr/share` Zusätzliche Daten, die die Funktionalität von Programmen erweitern oder ergänzen; z. B. Schriften, Logos, Skripte, ...

`/usr/share/doc` Sofern vorhanden: Dokumentation.

Befehle

Die Shell führt jede eingegebene Zeile als Befehl aus. Eine typische Befehlszeile beginnt mit dem Namen des Befehls gefolgt von optionalen Argumenten. Es gibt ein paar Sonderzeichen mit Spezialeffekten, außerdem bietet die Shell auch einfache Programmierkonstrukte wie Variablen und Schleifen, mit denen große Aufgaben (z. B. das Umbenennen von vielen Dateien) automatisiert werden können. Ein paar Beispiele:

```
help
man sh
ls -l /var/log >logfiles.txt
grep log logfiles.txt | wc
for i in *; do echo $i; done
for i in *.JPG; do mv $i $(basename $i JPG)jpg; done
```

Die Ausgabe eines Befehls wird normalerweise am Bildschirm angezeigt, kann aber (mit „`>`“) in eine Datei umgeleitet oder (mit einem sog. Pipe-Symbol „`|`“) als Eingabe für einen weiteren Befehl verwendet werden.

Übersicht der wichtigsten Befehle

`cd` Wechselt das Verzeichnis. Es können absolute Pfade angegeben werden, wie `cd /lib` oder relative `cd modules`. Als Besonderheit gibt es in *jedem* Verzeichnis einen Eintrag `..` (zwei Punkte), der das jeweils darüberliegende Verzeichnis meint. Also mit `cd ..` kommt man aus einem Verzeichnis heraus.

`ls` Zeigt den Inhalt des Verzeichnis an. Die Optionen `-a` (versteckte Dateien anzeigen) und `-l` (Details anzeigen) sind besonders wichtig.

`echo` Gibt alle Argumente auf dem Standardausgabegerät aus. – Für Umsteiger ist es vielleicht befremdlich, dass es für so eine einfache Aufgabe ein eigenes Programm gibt. Tatsächlich gehört `echo` aber zu den wichtigsten Tools, die auf absolut jedem Linuxsystem installiert sind.

`cat` Alle Argumente werden als Dateien aufgefasst und eine nach der anderen auf der Standardausgabe ausgegeben. Unter anderem praktisch um kleine Textdateien am Bildschirm anzuzeigen.

`more`, `less` Ähnlich wie `cat` aber mit dem Unterschied, dass nur maximal eine Bildschirmseite angezeigt wird. Mit der Leertaste kann man weiterblättern. `less` erlaubt außerdem das Zurückscrollen mit den Pfeiltasten.

`man` Auf Unixsystemen gibt es normalerweise zu jedem Programm eine manual page (Handbuchseite). `man` mit einem Befehlsnamen als Argument zeigt diese Handbuchseite an. Dies ist meist der schnellste Weg, um herauszufinden, welche Argumente ein Befehl versteht bzw. was eine bestimmte Option bedeutet. Beispiel: `man man`

`apropos` Jede manual page hat eine Kurzbeschreibung. Die Argumente von `apropos` sind Suchwörter. Alle passenden Kurzbeschreibungen werden ausgegeben. Das ist sehr praktisch, wenn man für eine bestimmte Aufgabe einen Befehl sucht, dessen Namen man vergessen hat. Die interessanten Befehle aus der ausgegebenen Liste kann man dann mit `man` genauer nachlesen. Beispiel: `apropos page`

`whatis` Gibt zu einem Befehl die Kurzbeschreibung aus.

ssh Über eine verschlüsselte Verbindung mit der Shell auf einem anderen Rechner im Netzwerk oder Internet verbinden. Z. B. `ssh root@192.168.0.1`

sudo Einen Befehl mit Administratorrechten ausführen. Es ist wesentlich sicherer als normaler Benutzer zu arbeiten und nur ausgewählte Befehle mit `sudo` zu starten anstatt immer als Administrator zu arbeiten.

sudoedit Eine geschützte Datei bearbeiten.

exit Die Shell beenden.

cp Kopieren von Dateien (Verzeichnisse mit `-r`).

mv Eine Datei (Verzeichnis) verschieben bzw. umbenennen.

rm Eine Datei löschen (für immer). Mit der Option `-r` werden Verzeichnisse rekursiv gelöscht. Mit `-f` werden auch schreibgeschützte Dateien gelöscht, ohne nachzufragen.

mkdir Verzeichnis anlegen.

chmod Zugriffsrechte einer Datei verändern. Unter Unix sind jeder Datei Lese-, Schreib- und Ausführungsrechte zugeordnet und zwar getrennt für den Besitzer einer Datei, die Gruppe des Besitzers und alle anderen.

mount Wie schon erwähnt gibt es unter Unix keine Laufwerksbuchstaben, sondern alle Dateien werden in einem gemeinsamen Hierarchie dargestellt. `mount` bindet den Inhalt eines neuen Datenträgers ein. Beispiel für eine Diskette:
`mount /dev/fd0 /media/floppy0`

grep Durchsucht die Eingabe nach einem Textmuster und gibt alle passenden Zeilen auf der Standardausgabe aus. Als Muster können reguläre Ausdrücke verwendet werden – siehe manpage.

gzip Das übliche Komprimierungstool unter Unix. Mit der Option `-d` werden bereits komprimierte Dateien wieder ausgepackt. `gzip` kann allerdings keine Archive erzeugen, dafür wird meist `tar` verwendet.

tar Das Standardtool um Archive zu erzeugen und verwalten. Es gibt viele Optionen, die wichtigsten sind `-x`, `-c`, `-f`, `-z`. Beispiel: `tar -xzf archiv.tgz`

ps Zeigt gerade am System laufende Prozesse an. Mit `-A` werden alle Prozesse angezeigt. Nützlich um zu sehen, ob ein Prozess überhaupt noch läuft und wieviel CPU-Zeit er verbraucht.

kill Einen Prozess beenden. Die Option `-9` erzwingt das sofortige Beenden, z. B. wenn der Prozess abgestürzt ist. Der Prozess wird als PID (siehe Ausgabe von `ps`) angegeben. Beispiel: `kill -9 1234` — technisch gesehen sendet `kill` ein sogenanntes Signal an den Prozess. Manche Signale können auch durch Tastenkombinationen ausgelöst werden, allerdings ausgerechnet das Signal `-9`, das sofortiges Beenden erzwingt, nicht.

Tastenkombinationen

Je nach Arbeitsumgebung gibt es eine Vielzahl verschiedener Tastenbelegungen. Hier wird nur auf die unmittelbar für die Shell wichtigsten eingegangen:

Strg-C Sendet an den momentan aktiven Prozess ein sog. Interrupt-Signal. Das hat bei den meisten Programmen eine ähnliche Wirkung wie wenn es mit `kill` beendet wird.

Strg-Z Sendet an den momentan aktiven Prozess ein Stop-Signal. Dadurch wird die Ausführung des Programms angehalten (und der Benutzer zurück in die Shell geworfen) ohne den Prozess wirklich zu beenden. Der Prozess kann dann z. B. mit dem Befehl `bg` im Hintergrund fortgesetzt werden.

Strg-S Bildschirmausgabe anhalten. Das ist auf modernen Systemen selten nützlich, kann aber verwirren, wenn es unabsichtlich gedrückt wurde und der Computer scheinbar nicht mehr reagiert. Mit Strg-Q kann die Sperre wieder aufgehoben werden.

Strg-R Die meisten Shells haben eine Befehlshistory, über die z. B. mit den Pfeiltasten die zuletzt verwendeten Befehle wieder aufgerufen werden können. Mit Strg-R kann in dieser History rückwärts inkrementell gesucht werden. Einfach ausprobieren und anfangen einen alten Befehl eingeben ...

Tab Moderne Shells können Befehle selbst vervollständigen.

Environment

Unter Unix bekommt jeder Prozess beim Start ein sogenanntes Environment, das ist eine Liste von Variablen, die das Programm ähnlich wie eine Konfigurationsdatei verwenden kann um Einstellungen zu lesen. Es gibt also (bei manchen Programmen) zwei Möglichkeiten eine Konfiguration vorzunehmen – über Konfigurationsdateien *und* das Environment. Das kann manchmal zu Verwirrung führen ...

Der Befehl `env` zeigt das aktuelle Environment an. Mit dem Befehl `export VAR=value` wird eine Variable für die Dauer einer Shell-Sitzung gesetzt. Soll nur ein einziger Befehl mit einer bestimmten Variable ausgeführt werden, so ist die Syntax `LANG=C man env` um z. B. statt der deutschen die englische manpage von `env` zu lesen, praktisch.

Welche Umgebungsvariablen ein bestimmtes Programm auswertet, ist nicht allgemein festgelegt, steht aber in der manpage. Deshalb hier nur eine Liste der Meistverwendeten:

PATH Wird von der Shell verwendet um die Programme zu finden. Diese Variable ist einfach eine Liste von Verzeichnissen, in denen nach Programmen gesucht werden soll. Soll ein Programm, das in einem anderen Verzeichnis liegt, ausgeführt werden, dann muss der Pfad beim Befehlsnamen explizit mit angegeben werden.

EDITOR Es gibt Befehle (z. B. `sudoedit`), die einen interaktiven Editor starten. Sind mehrere Editoren installiert, so kann ein bestimmter oft über diese Variable ausgewählt werden.

TERM Informationen über das Terminal, für das die Ausgabe formatiert werden soll. Das ist wichtig, damit man, wenn man sich mit `ssh` auf einen anderen Rechner verbindet, trotzdem eine passende Ausgabe für das *eigene* Terminal bekommt. Normalerweise funktioniert das alles automatisch. Folgende Werte sind üblich: `xterm`, `linux`, `ansi`, `vt100`, ...

LANG sagt welche Spracheinstellungen verwendet werden sollen. Wichtig ist das vor allem, falls ein Programm mit deutschen Einstellungen fehlerhaft funktioniert. Dann hilft es oft, temporär `LANG=C` zu setzen.

Anregungen (zum Spielen und Lesen)

Was machen die folgenden Befehle (sofern sie installiert sind): `pwd`, `free`, `df`, `rmdir`, `passwd`, `date`, `info`, `killall`, `uname`, `uptime`, `chown`, `find`, `whoami`, `alias`, `file`, `fg`

Außerdem noch ein paar interessante/lustige Dateien: `/dev/zero`, `/dev/urandom`, `/dev/null`, `/proc/partitions`, `/proc/cpuinfo`, `/etc/profile`, `/etc/environment`, `/etc/sudoers`, `/usr/local/`