Die fabelhafte Welt der Shell

Während sich Anfänger mit ihr kaum anfreunden können, mögen viele Nutzer nicht mehr auf sie verzichten: Die Eingabeaufforderung (»Shell«) unter Unix.

Fortgeschrittene Nutzer schätzen die Geschwindigkeit, mit der sie auch komplexe Aufgaben so erledigen können - auch von zuhause aus, über eine langsame Modem-Verbindung. Aber auch lokal bevorzugen sie meist die Shell gegenüber den »komfortablen« Benutzeroberflächen wie KDE und GNOME.

Hier eine kleine Einführung in die *fabelhafte Welt der Shell*.

Neben einigen wichtigen Prinzipien könnt ihr hier auch gleich einen Haufen hilfreicher Kommandos kennenlernen.

Mithilfe gefragt!

Die meisten Studenten haben vor ihrem Studium kaum Kontakt zu Unix gehabt. Viele tun sich schwer, sich an die Arbeit mit Unix zu gewöhnen. Dazu würden wir in der Fachschaft gerne eine Sammlung von Informationen rund um Unix zusammenstellen - wir sind dabei aber weniger an Erklärungen von sogenannten »Freaks« interessiert - viel mehr hätten wir gerne gewusst, welche Infos für Euch besonders hilfreich waren, was die wichtigsten Punkte waren, die ihr erst herausfinden musstet.

Daher wäre es Klasse, wenn ihr uns solche Infos per E-Mail schickt an

`erich@fachschaften.uni-muenchen.de`

damit wir sie auf hier einbauen können.

Dateien, Dateien, nichts als Dateien

Unter Unix wird alles als Datei aufgefasst - inklusive der Eingabe per Tastatur und der Ausgabe auf den Bildschirm.

Weil aber alles eine Datei ist, so kann man nun auch die Eingabe eines Programmes aus einer Datei lesen, die Ausgabe in eine andere Datei schreiben - oder die Ausgabe eines Programmes als Eingabe eines anderen Programmes verwenden (sogenanntes »Pipelining«).

Die Eingabe solcher Umleitungen ist ganz einfach (daran erkennt man wie wichtig sie ist): `prog < eingabedatei` liest aus einer Datei, `prog > ausgabedatei` schreibt in eine, `prog1 | prog2` verwendet die Ausgabe von `prog1` als Eingabe von `prog2` zu verwenden. `>` überschreibt die alte Datei, will man an die Datei anhängen verwendet man `>>`. Das wars auch schon, eigentlich nur 3 Zeichen die man lernen muss!

Baukasten-Prinzip

Man kann sich das ganze wie einen Baukasten vorstellen. Aus vielen kleinen Programmen (die oft nicht mehr können als eine Datei auszugeben, oder die Zeilen in der Datei zu zählen) baut man sich den gewünschten Vorgang zusammen.

Viele Programme in der Unix-Welt arbeiten nämlich wie Filter. Um dies zu veranschaulichen wird hier eine typische Verkettung von Programmen angegeben (das Zeichen `|` nennt man übrigens »Pipe«, weil man die Ausgabe eines Programmes sozusagen mit einem Rohr an den Eingang eines anderen leitet, auf der deutschen Tastatur erreichbar über `AltGr+<`)

```
zcat skript.ps.gz | psbook \
| psnup -2 | gzip > skript-buchformat.ps.gz
```

So, das schaut jetzt schon ziemlich kryptisch aus... Ist aber eigentlich ganz einfach:

Das erste Programm, `zcat`, kann eine gzip-Komprimierte Datei (Endung `.gz`) entpacken und ausgeben. Der Name kommt von »zipped `cat`«, wobei »`cat`« einfach die Kurzform von »concatenate« ist – damit kann man auch mehrere Dateien hintereinander ausgeben und so zusammenhängen.

Diese Ausgabe (das Skript ist eine Postscript-Datei `.ps`) wird jetzt vom Programm `psbook` verarbeitet. Dieses Programm macht nichts weiter, als die Seiten so umzusortieren, wie man sie für das Drucken eines Buches braucht. Das Programm `psnup` kann sogenannte n-up-Formate (Fachausdruck aus dem Layout-Bereich für n Seiten auf einem Blatt, die alle gleich ausgerichtet sind. So kann man papiersparender drucken...) erzeugen.

Die Ausgabe daraus wird jetzt wieder mit dem Programm `gzip` komprimiert und in die Datei `skript-buchformat.ps.gz` abgespeichert.

Wozu jetzt also dieses Kommando?

Ganz einfach, `skript.ps.gz` war beispielsweise ein Skript einer Vorlesung auf DIN A4. Das Ergebnis ist das Skript mit zwei A5-Seiten auf einer A4-Seite, und so umsortiert, dass man den Ausdruck einfach in der Mitte falten, und wie ein Heft lesen kann (dazu muss nämlich die erste und die letzte Seite auf das selbe Blatt gedruckt werden, diese Umordnung der Seiten hat `psbook` vorgenommen). Möchte man nur zwei Seiten nebeneinander, mit der normalen Reihenfolge (um es beispielsweise in einen Ordner abzuheften), so lässt man das `psbook` Kommando weg. Aus Platzgründen (Postscript wird manchmal recht groß) soll das Ergebnis gleich wieder `gzip`-komprimiert gespeichert werden.

Drucken kann man das ganze dann mit `zcat skript-buchformat.ps.gz | lpr`.

Dabei kommt `lpr` übrigens vom Ausdruck »line printer«. Wenn ihr etwas drucken wollt, wandelt ihr es am besten in Postscript und druckt es dann mit `lpr`.

Parameter

Unix-Shell-Kommandos haben Parameter, die sich meist in zwei Klassen einteilen lassen: es gibt Optionsparameter (diese werden mit `-` oder mit `--` eingeleitet) und Dateinamen, ein Dateiname kann hierbei auch eine Pfadangabe enthalten (z.B. `unterverzeichnis/datei.txt`)

Optionsparameter sind beispielsweise bei `ls` die Parameter `-l` und `-a` (oder gleichbedeutend `--all`: alle Dateien anzeigen).

Die meisten Shells bieten einem zur schnelleren Eingabe von Dateinamen die sogenannte »Tab-Completion« an. Dazu gibt man den Anfang des Dateinamens ein, drückt die Tabulator-Taste (links neben Q) und wenn der Anfang eindeutig war, wird der Dateiname vervollständigt (mit erneutem Drücken von Tab kann man sich die Möglichkeiten anzeigen lassen, falls es nicht eindeutig war).

Eine Sonderrolle spielt der Dateiname »-«. Dieser Kennzeichnet die Standard-Eingabe bzw. Standard-Ausgabe (also normalerweise Tastatur bzw. Bildschirm) und kann in vielen Programmen verwendet werden.

Das Kommando von vorher könnte man z.B. auch so schreiben:

```
zcat skript.ps.gz | psbook - - \
| psnup -2 - - | gzip - - > skript-buchformat.ps.gz
```

(`cat` und `zcat` geben immer auf die Standardausgabe aus)

Zentrale Kommandos

Um mit der Shell arbeiten zu können, muss man natürlich ein paar Kommandos kennen. Alle kann man nicht kennen (es gibt oft mehrere tausend!), aber bei einem Satz von vielleicht 20 Stück sollte man nicht mehr nachlesen müssen. Dazu gehören vor allem `cd verzeichnis` (»change directory«) um in ein anderes Verzeichnis (einen anderen Ordner) zu wechseln, `cd ..` um ein Verzeichnis nach oben zu gehen, `pwd` um sich den aktuellen Pfad anzeigen zu lassen (»print work directory«) und `cd` um in das persönliche Verzeichnis (Homedirectory) zu springen.

Mit `ls` – das kommt von »list« – kann man sich den Inhalt des aktuellen Verzeichnisses anzeigen, mit `ls verzeichnis` den eines anderen Verzeichnisses. Möchte man detailliertere Information (Zugriffsrechte, Eigentümer, Änderungsdatum, Größe) erhalten, so geht das mit Hilfe von `ls -l`.

Neue Verzeichnisse legt man mit `mkdir` an, löschen kann man sie mit `rmdir`, wenn sie leer sind.

Verschieben und Uumbenennen kann man Dateien mit dem Kommando `mv`, Löschen mit dem Kommando `rm`.

Textdateien kann man sich mit `more` oder mit `less` Seitenweise anschauen.

`file` kann einem bei den meisten Dateien anzeigen, in welchem Format sie abgespeichert sind.

Es gibt viele weitere wichtige Kommandos – aber welche man selbst häufig nutzt ist eine Geschmacksfrage.

Arbeiten auf anderen Rechnern

Oft kommt es vor, dass man an einem Rechner sitzt, aber auf Dateien oder Programme zugreifen muss, die auf einem anderen Rechner liegen. Hier bietet sich die Verwendung des stark verschlüsselnden SSH-Protokolls an. Mit dem Kommando `ssh benutzer@rechner` kann man sich auf einem anderen (Unix-)Rechner anmelden und dort Programme starten - verwendet man `ssh -X benutzer@rechner`, dann sogar graphische. Unter Windows gibt es dafür das kostenlose Programm »putty« (einfach bei Google nach putty suchen), das sozusagen ein xterm und ssh für Windows kombiniert (allerdings »ohne -X«).

Um also z.B. zu einem Rechner in der Informatik eine Verbindung aufzubauen gebt ihr auf einem Unix-Rechner folgendes ein:
`ssh benutzername@remote.cip.informatik.uni-muenchen.de`.

Analog kann man auch Dateien von einem Rechner auf einen anderen kopieren: »`scp dateien benutzer@rechner:`« um auf den anderen Rechner zu kopieren, »`scp benutzer@rechner:pfad/datei .`« um Dateien vom anderen Rechner zu holen (der Pfad ist relativ zum eigenen Verzeichnis).

Für Windows gibt es hier das ebenfalls kostenlose Programm »WinSCP« (wiederum per Google zu finden), das auf Putty aufsetzt.

Dateien komprimieren und archivieren

Oft (z.B. um ein Numerik-Übungsblatt abzugeben) muss man mehrere Dateien zu einem Archiv zusammenfassen. Das übliche Kommando dafür ist `tar` - es gibt aber auch `zip` (Windows) und `jar` (Java); `tar` hat leider eine relativ ungewöhnliche Syntax (und `jar` verwendet die selbe Syntax).

Um mehrere Dateien / Ordner zu einem komprimierten Archiv zusammenzufassen verwendet man das folgende Kommando: `tar czf archiv.tar.gz dateien...`, um das Archiv zu entpacken das Kommando `tar xzf archiv.tar.gz`. Diese Kommandos setzen sich folgendermaßen zusammen: `tar` is das eigentliche Programm. Der Buchstabe `c` steht für »create«, `x` steht für »extract« - also den gewünschten Vorgang. `z` gibt an, dass die Dateien mit gzip komprimiert werden sollen, und `f archiv.tar.gz` gibt den Ausgabedateinamen an. Man kann auch noch `vv` angeben, dann zeigt `tar` an, welche Dateien es archiviert.

Statt dem ersten Kommando könnte man also auch
`tar cz dateien... > archiv.tar.gz` schreiben oder
`tar c dateien... | gzip > archiv.tar.gz`.

Besser drucken

Ein Weg »besser« zu Drucken wurde oben schon gezeigt – mit zwei Seiten pro Blatt durch »`zcat datei.ps.gz | psnup -2 | gzip > datei2.ps.gz`«.

Jetzt sind nur noch zwei Probleme zu lösen: wie kann man sich das Ergebnis einmal anschauen vor dem Ausdruck – und wie erzeugt man so eine `.ps.gz` Datei?

Das erste Problem ist einfach: »`gs datei2.ps.gz`« zeigt die Datei an, wie sie ausgedruckt würde.

PostScript-Dateien (dafür steht `.ps`), die der Drucker direkt versteht, können viele Programme direkt erzeugen. Wenn ihr beispielsweise im Mozilla oder Netscape

Browser oder in irgend einem KDE-Programm auf Drucken geht, könnt ihr in eine Datei drucken – und schon habt ihr eine unkomprimierte PostScript-Datei (nachträglich komprimierbar per `gzip Datei.ps`).

Und – wenn ihr Informatik studiert werdet ihr das oft brauchen – einen Quellcode wandelt ihr am besten per »`a2ps --pretty-print datei.java -o druck.ps`« um: »`--pretty-print`« sorgt dafür dass die Ausgabe besser lesbar wird.

Kommando-Vielfalt

Ein Problem, vor dem die meisten früher oder später stehen ist die Vielzahl der Kommandos. Auf einem typischen Linux-System existieren bis zu 6000 Kommandos (wobei man die meisten davon nie direkt aufrufen wird...) - wie soll man sich da noch zurecht finden?

Eigentlich reicht es, wenn man hier zwei Kommandos kennt: `apropos` und `man`.

Mit `apropos suchbegriff` kann man die Beschreibungen der Kommandos durchsuchen, und so die richtigen Kommandos finden; mit `man kommando` kann man sich dann eine Anleitung zu dem jeweiligen Kommando anzeigen lassen. (Mit `/begriff` kann man übrigens in der Anleitung suchen, das ist sehr hilfreich bei langen Anleitungen!)

Oft ist es dann am einfachsten, man macht ein zweites Fenster auf, schaut sich dort die entsprechende »man-page« an (und sucht dort nach den Beispielen...).

Die so gerlernten Kommandos notiert man sich am besten gleich in einer eigenen Datei `wichtige-unix-kommandos.txt` - vielleicht braucht man sie ja mal wieder. Oder ein Kommilitone hat mal das selbe Problem.

Nie aufgeben

»There is more than one way to do it« – man muss nur herausfinden wie. Wenn man gar nicht mehr weiter weiß – oder sich von der Länge der »man pages« abschrecken lässt – kann man ja immernoch einen Kommilitonen fragen.

Und nicht vergessen:
Schickt uns per E-Mail, was ihr wichtiges über Unix gelernt habt. Danke!

Erich Schubert