Einige Kommandozeilenbefehle für Linux

Befehle zur Arbeit mit Verzeichnissen

`pwd`	:	Pfadname des aktuellen Verzeichnisses anzeigen
`mkdir` *dirname*	:	Verzeichnis *dirname* erzeugen
`rmdir` *dirname*	:	Verzeichnis *dirname* löschen (nur wenn es leer ist)
`ls`	:	Inhalt des aktuellen Verzeichnisses (Dateinamen) auflisten
`ls` *dirname*	:	Inhalt des angegebenen Verzeichnisses
`ls *.o`	:	alle Objektfiles im aktuellen Verzeichnis auflisten
`ls -l`	:	ausführliches Inhaltsverzeichnis (Dateiattribute), auch: `ll`
`ls -ltr`	:	sortiert nach Zeitpunkt der letzten Änderung (statt alphabetisch)
`cd` *dirname*	:	ins angegebene Verzeichnis wechseln
`cd ..`	:	im Verzeichnisbaum eine Stufe zurück
`mv` *oldname newname*	:	Umbenennen eines Verzeichnisses (oder einer Datei)
`du -sh .`	:	Anzeige, wieviel Speicher vom aktuellen Verzeichnis und allen Unterverzeichnissen belegt wird

Befehle zur Arbeit mit Dateien

`cat` *file*	:	Inhalt der Datei anzeigen
`more` *file*	:	Inhalt größerer Dateien seitenweise anzeigen (zum „Blättern“)
`less` *file*	:	etwa dasselbe wie `more`
`head -`*n file*	:	nur die ersten *n* Zeilen der Datei *file* anzeigen
`tail -`*n file*	:	nur die letzten *n* Zeilen der Datei *file* anzeigen
`tail -f` *file*	:	nach Anzeige der Datei wartet das Programm und zeigt jeweils (durch anderes Programm) neu hinzukommende Zeilen an; muss mit `Ctrl-C` abgebrochen werden.
`mv` *oldname newname*	:	Datei umbenennen
`mv` *files... dirname*	:	Datei(en) in das Verzeichnis *dirname* verschieben
`cp` *file newfile*	:	Datei kopieren (unter neuem Namen)
`cp` *files... dirname*	:	Datei(en) ins angegebene Verzeichnis kopieren
`ln -s` *pfad/file newfile*	:	Zugriff auf Originaldatei über anderen Namen oder anderen Pfad (symbolischer Link)
`rm` *files...*	:	Datei(en) löschen (kein „Papierkorb“!)
`chmod` *modus files...*	:	Zugriffsrechte für Datei(en)/Verzeichnisse ändern, z.B.
`chmod a+r` *file*	:	für alle lesbar (und damit kopierbar)
`chmod go-w` *file*	:	für Nutzergruppe und andere Schreibverbot
`chmod ug+rx` *file*	:	für Eigentümer und Nutzergruppe les- und ausführbar (Programm) bzw. durchsuchbar (Verzeichnis)

Befehle zum Editieren von Textdateien (Quelltexte, Makefile, ...)

`nedit` *file* `&`	:	einfache Bedienung, Standardfunktionalität, Syntax-Highlight möglich
`kedit` *file* `&`	:	einfache Bedienung, sparsamste Funktionalität
`xemacs` *file* `&`	:	anspruchsvoll, sehr viele Funktionen, auch zum Einbinden von Anwendungsprogrammen, Syntax-Highlight möglich
`vi` *file*	:	benötigt keine grafische Oberfläche, d.h. Bedienung in der Befehls-Konsole ohne Extra-Fenster, relativ kompliziert, aber sehr leistungsstark für spezielle Edit-Funktionen,

Bem.: Das Zeichen `&` nach einem Befehl bewirkt seine Abarbeitung *im Hintergrund*, d.h. ohne auf dessen Ende zu warten.

Dateien archivieren (Ein- und Auspacken zum besseren „Transport“)

`tar cvf` *`file`*`.tar` *`dirname`*	:	alle Dateien und Unterverzeichnisse des angegebenen Verzeichnisses in eine Archivdatei kopieren
`tar xvf` *`file`*`.tar`	:	Dateien aus Archivdatei ins aktuelle Verzeichnis extrahieren (mit Unterverzeichnissen)
`gzip` *`file`*`.tar`	:	Datei komprimieren (*`file`*`.tar` wird ersetzt durch *`file`*`.tar.gz`)
`gunzip` *`file`*`.tar.gz`	:	Datei wieder dekomprimieren

mit der GNU-Version von `tar` auch:

`tar cvzf` *`file`*`.tar.gz`	:	`tar` und `gzip` in einem
`tar xvzf` *`file`*`.tar.gz`	:	`gunzip` und `tar` oft wird *`file`*`.tgz` statt *`file`*`.tar.gz` benutzt

Da bei `tar` die Dateien ihr Änderungsdatum beibehalten (im Gegensatz zu `cp`), wird zum Kopieren ganzer Verzeichnisse auch gern folgende Konstruktion benutzt:

```
cd todir
(cd fromdir; tar cf - .) | tar xvf -
```

Hier steht der spezielle Dateiname „`-`“ für die Standardausgabe bzw. Standardeingabe und der Operator „`|`“ leitet die Ausgabe des linken Kommandos in die Eingabe des rechten Kommandos um.
(So könnte man statt `tar xvf file.tar` auch schreiben: `cat file.tar | tar xvf -`)

Einsparung lästiger Tipparbeit

- durch Verwendung der Tabulatortaste:
  Beim Eintippen eines Dateinamens wird durch TAB automatisch ergänzt, falls die ersten Buchstaben den Namen schon eindeutig bestimmen. Anderenfalls werden alle möglichen Ergänzungen angezeigt.
  Letzteres funktioniert nur, wenn diese Option auch eingeschaltet ist
  (dazu dient das Kommando: `set autolist`)

- durch Platzhalter für Dateinamen (Wildcards):
  `?` für ein beliebiges Zeichen, `*` für beliebig viele beliebige Zeichen (Ausnahme: kein `.` am Anfang eines Dateinamens),
  Zeichengruppen in `[ ]` (für ein beliebiges der angegebenen Zeichen),

  Beispiele:

`rm /tmp/*`	:	alle Dateien im Verzeichnis `/tmp` löschen, deren Name nicht mit `.` beginnt.
`rm .*`	:	alle Dateien löschen, die mit `.` beginnen (**nie** im eigenen Home-Verzeichnis ausprobieren!)
`ls *.o`	:	listet alle Objektfiles im aktuellen Verzeichnis auf
`ls *.[fFc]`	:	alle Dateien, deren Endung `.f, .F` oder `.c` ist
`ls [a-dX-Z]*`	:	alle Dateien, die mit `a, b, c, d, X, Y,` oder `Z` beginnen

  Warnung: Nach dem Eintippen eines Befehls mit einem Blick auf den Bildschirm nochmal kontrollieren, was man eingetippt hat!
  Die Wirkung eines Tippfehlers kann verheerend sein, z. B.

  `rm * ~` (statt `rm *~`)

  Wenn die Fehlermeldung erscheint, dass `'~'` nicht gelöscht werden kann, sind bereits alle Dateien (`'*'`) unwiderruflich gelöscht.

Suchen nach Dateien

`find . -name file -print`
sucht im aktuellen Verzeichnis (und Unterverzeichnissen) nach allen Dateien mit dem angegebenen Namen und zeigt den Pfad dazu an

`find . -name '*.f' -mtime +7 -ls`
sucht alle Dateien mit der Endung `.f`, die vor mehr als 7 Tagen letztmalig verändert wurden und zeigt diese an wie das Kommando `ls -l`; mit `-7` entsprechend alle Dateien, die innerhalb der letzten 7 Tage geändert wurden.

`find . -name '*~' -exec rm {} \;`
sucht alle Dateien, die auf `~` enden (z.B. Backup-Dateien des Editors) und führt für jede das `rm`-Kommando zum Löschen aus.

`find . -name '*.o' -ok rm {} \;`
Bei jeder gefundenen Datei mit der Endung `.o` wird der Nutzer gefragt, ob sie gelöscht werden soll oder nicht.

Wenn man dem System nicht so recht traut, geht's auch so (etwas vorsichtiger):

`find . -name '*.o' -print`
Anzeige der gefundenen Dateien — und wenn man diese dann löschen will: `rm ‘!!‘`
das bedeutet: letzten Befehl noch einmal ausführen (`!!`) und dessen Ausgabe als Argument in die Kommandozeile von `rm` schreiben (`rm ‘befehl‘`)

Beachte den Unterschied auf der Tastatur zwischen ‘ ‘ und ’ ’ (Apostroph) !

Weitere hilfreiche Befehle

`man befehl`	:	Online-Manual zum angegebenen Unix-Kommando (oder C-Unterprogramm o.a.)
`date`	:	Ausgabe von Datum und Uhrzeit
`cal`	:	Ausgabe eines Kalenders (aktueller Monat)
`gfortran -o myexe myprog.f`	:	Übersetzen der Quelldatei *myprog*`.f` und (im Erfolgsfall) ausführbare Datei *myexe* erzeugen.
`gcc -o myexe myprog.c`	:	(analog für eine C-Quelldatei)
`./myexe`	:	das im aktuellen Verzeichnis stehende Programm *myexe* ausführen.
`which befehl`	:	zeigt an, in welchem Verzeichnis der angegebene Befehl gefunden wird (bei Benutzung der aktuellen Suchpfad-Einstellungen), bzw. `'shell built-in command'` für solche Befehle wie `echo, foreach, if, which, ...`

Ein- und Ausgabeumlenkung

Jedes Programm (jeder Befehl) kann auf zwei spezielle Dateien zugreifen, die Standardeingabe (`stdin`) und die Standardausgabe (`stdout`), in Fortranprogrammen die Dateinummern `5` und `6`. Diese Dateien entsprechen im Normalfall der Tastatureingabe und der Bildschirmausgabe. Um so ein Programm wiederholt oder im Hintergrund zu testen, ohne ständig die entsprechenden Eingaben von Hand machen zu müssen, oder die Ergebnisse *live* am Bildschirm zu verfolgen, kann man diese Ein- und Ausgaben auf Dateien *umlenken*. Alle erforderlichen Tastatureingaben schreibt man vorher in eine Datei, z.B. `eingabe.txt`, und alle Ausgaben des Programms lässt man sich in eine Datei schreiben, z.B. `ausgabe.txt`, um sie später auszuwerten. Die Ein-/Ausgabeumlenkung erfolgt dann durch

```
./myexe <eingabe.txt >ausgabe.txt
```