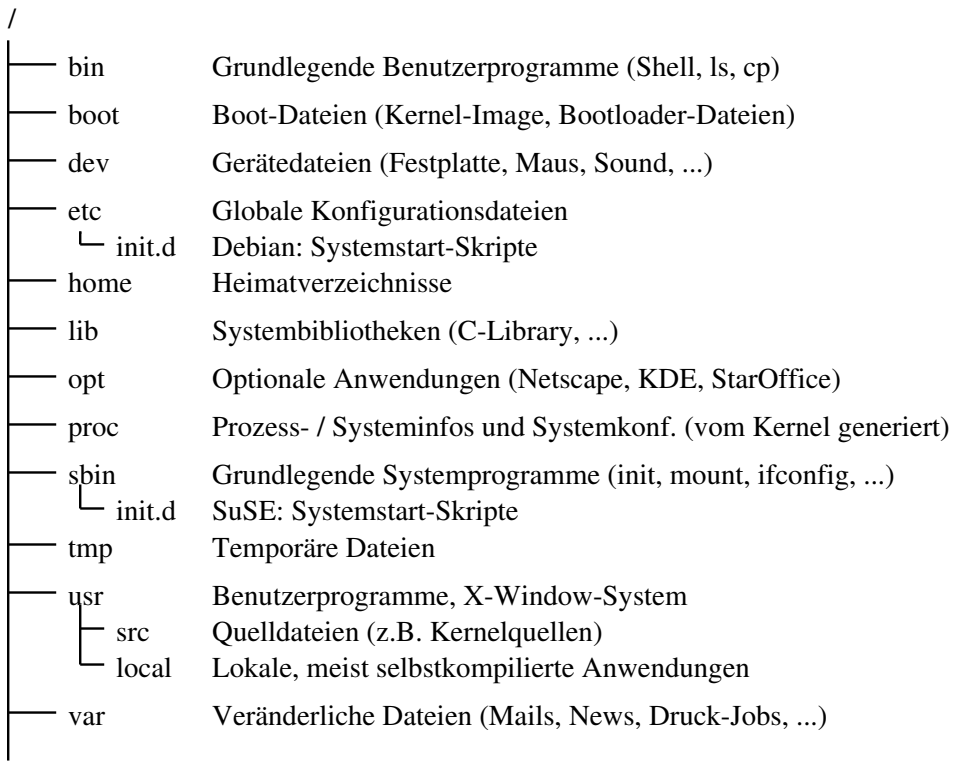
Wichtige Unix-Befehle

`man` *befehl*	Zeigt die Anleitung (Manual-Page) zu *befehl* an
`ls`	Gibt den Verzeichnisinhalt aus
`ls -l`	Gibt die Rechte(Permissions) der Dateien aus
`ls -a`	Zeigt auch versteckte Dateien an
`cd` *verzeichnis*	Wechselt in das Unterverzeichnis *verzeichnis*
`cd ..`	Wechselt zurück in das übergeordnete Verzeichnis
`mkdir` *verzeichnis*	Legt das Unterverzeichnis *verzeichnis* an
`rmdir` *verzeichnis*	Löscht das leere(!) Unterverzeichnis *verzeichnis*
`touch` *datei*	Erstellt die leere Datei *datei*
`cp` *datei1 datei2*	Kopiert die Datei *datei1* nach *datei2*
`mv` *datei1 datei2*	Verschiebt die Datei *datei1* nach *datei2*
`rm` *datei*	Löscht die Datei *Datei*
`rm -r` *verzeichnis*	Löscht das Verzeichnis *verzeichnis* rekursiv! ACHTUNG!
`ln -s` *datei1 datei2*	Legt einen symbolischen Link Namens *datei2* auf *datei1* an
`find . -name` *datei*	Nach der Datei *datei* im aktuellen Verzeichnis (.) suchen
`chown` *user datei*	Gibt die Rechte an der Datei *datei* dem Benutzer *user*
`chmod a+x` *datei*	Macht die Datei *datei* für jeden (a) ausführbar (x) `a`=Alle (all), `u`=Benutzer (user), `g`=Gruppe (group) `r`=Lesen (read), `w`=Schreiben (write), `x`=Ausführen (exec)
`cat` *datei*	Gibt die Datei *datei* auf dem Bildschirm aus
`less` *datei*	Gibt die Datei *datei* Seitenweise aus
`grep "string"` *datei*	Nach "`string`" in der Datei *datei* suchen
`joe` *datei* `emacs` *datei* `vi` *datei*	Editiert die Datei *datei*
`ps`	Zeigt die laufenden Prozesse des aktuellen Benutzers an
`ps ax`	Zeigt alle laufenden Prozesse an
`kill` *PID*	Fordert den Prozess mit der ID *PID* auf, sich zu beenden
`kill -9` *PID*	Tötet den Prozess mit der ID *PID*
`passwd / yppasswd`	Ändert das Passwort (Nicht am HGF probieren - NT/Linux sync.!)
`chsh`	Login-Shell ändern
`mutt, pine, mail`	Mails lesen und schreiben
`echo "Hallo Welt"`	`Hallo Welt` auf dem Bildschirm ausgeben
`read variable`	Liest eine Eingabe von der Tastatur in die Variable *variable* ein
`befehl &`	Startet den Befehl *befehl* im Hintergrund
`ALT-F`*n*	Schaltet auf die Konsole Nummer *n* um
`exit, logout, CTRL-D`	Beendet die aktuelle Sitzung

Der Verzeichnisbaum



Grundlegende Shell-Skript-Programmierung

Shell-Skripte sind eine Reihe von Unix-Befehlen in Textdateien. Diese werden von einem in der ersten Zeile eines Skripts genannten Interpreter (meist bash) ausgeführt. Ein Shell-Skript beginnt deshalb meist mit der Zeile:

`#!/bin/sh` *Dabei zeigt #! dem Kernel, dass es sich um ein Skript handelt, /bin/sh ist der Pfad zur Shell (Interpreter)*

Um ein solches Skript auszuführen, müssen die Rechte des Skripts auf "ausführbar" gesetzt werden. Dies erreicht man z.B. mit `chmod +x skript`.

Ein-/Ausgabeumleitung

Um die Ausgabe eines Programms in eine Datei umzuleiten, schreibt man: `befehl > datei`
Um diese Datei einem anderen Programm als Eingabe zu geben, gilt dann: `befehl < datei`
Wenn man die Ausgabe eines Programms direkt in die Eingabe eines anderen Programs umleiten will, sollte man das mit der Pipe (|) machen: `befehl1 | befehl2`

Grundsätzlich muss bei der Umleitung der Ausgabe beachtet werden, dass es 2 Ausgabekanäle gibt:

Die Standardausgabe	Die Standardausgabe wird mit `>` oder auch mit `1>` umgeleitet und enthält die Ausgabe des Programms ohne Fehlermeldungen
Der Standarderror	Der Standarderror-Kanal ist zur Ausgabe von Fehlermeldungen vorgesehen. Auch wenn die Standardausgabe umgeleitet wird, bleibt der Standarderror hiervon unberührt. Ihn kann man explizit mit `2>` umleiten. Falls man ihn an die gleiche Stelle wie die Standardausgabe umleiten will, bietet sich die Konstruktion `2>&1` an.

Variablen

Variablennamen können aus Klein-, Großbuchstaben und Zahlen bestehen. Um einer Variable einen Wert zuzuweisen, reicht es, z.B. `xyz="Hallo"` zu schreiben. Um auf eine Variable allerdings wieder zugreifen zu können, muss man dem Namen ein Dollarzeichen ($) voranstellen und kann ihn (um Portabilität zwischen verschiedenen Unixen zu gewährleisten) auch in geschweifte Klammern einschliessen:
`echo ${xyz}` *Ausgabe: Hallo*

Anführungszeichen

Die verschiedenen Typen von Anführungszeichen haben in der Shellprogrammierung eine besondere Bedeutung. Hier jeweils ein Beispiel (Voraussetzung ist, dass die Variable `variable` den Wert "`who am i`" enthält):

`echo Variable: "$variable"`
Ausgabe:
`Variable: who am i`
	Die doppelten Anführungszeichen (`"`) packen Text zu einem Block zusammen und erlauben die Auswertung enthaltener Variablen

`echo Variable: '$variable'`
Ausgabe:
`Variable: $variable`
	Die einfachen Anführungszeichen (`'`=Tick) packen Text zu einem Block zusammen, erlauben aber die Auswertung enthaltener Variablen *nicht*

`echo Variable: `$variable``
Ausgabe:
`Variable: stargo!michael...`
	Die auf den Kopf gestellten Anführungszeichen (`` ` ``=Backtick) führen den enthaltenen Text als Befehl aus, und übergeben die Ausgabe an den Aufrufer

Berechnungen

Um in der Shell Berechnungen durchzuführen, muss die Rechnung in `$[...]` eingeschlossen werden. So weist z.B. der Ausdruck `z=$[$y+3]` der Variable z den Wert der Variable y mit 3 addiert zu. Ein paar Beispiele für Berechnungen:

```
echo $[123+321]   444
echo $[2+3*5]      17
echo $[3/2]         1
```

Operatoren

`=`	gleich	`-gt`	grösser
`!=`	ungleich	`-lt`	kleiner

Bedingungen

Syntax: `if [ "`*wert/$variable*`" `*operator*` "`*wert/$variable*`" ]; then` ... `[else ...] fi`
oder: `test "`*wert/$variable*`" `*operator*` "`*wert/$variable*`" && `*befehl-wahr* `||` *befehl-falsch*

Wenn der übergebene Ausdruck wahr ist, werden die Befehle im `then`-Teil bzw. *befehl-wahr* ausgeführt. Falls der Ausdruck falsch ist, wird der evtl. vorhandene `else`-Teil bzw. *befehl-falsch* ausgeführt.
Beispiel:

```
if [ "$DISPLAY" != "" ]; then
   echo "Sie benutzen das X-Window System. Öffne ein neues Terminal..."
   xterm &
else
   echo "Bitte rufen sie dieses Skript unter X auf."
fi
```

Schleifen

Syntax: `for` *variable* `in` *Liste*`; do ... done`

Der Variable *variable* werden nacheinander alle Werte aus der Liste zugewiesen und dabei jedesmal der Programmcode zwischen `do` und `done` ausgeführt.
Beispiel: (Gibt nacheinander *Zahl 1*, *Zahl 3*, *Zahl 5* und *Zahl 7* aus.)

```
for i in 1 3 5 7; do
  echo "Zahl $i"
done
```

Bedingte Schleifen

Syntax: `while [ "`*wert/$variable*`" `*operator*` "`*wert/$variable*`"]; do` ... `done`

Die Schleife wird wiederholt, solange der übergebene Ausdruck wahr ist.
Beispiel: (Gibt nacheinander *Zaehler 1*, *Zaehler 2*, ..., *Zaehler 5* aus)

```
zaehler=1
while [ "$zaehler" -lt "6" ]; do
  echo "Zaehler: $zaehler"
  zaehler=$[$zaehler+1]
done
```