Wichtige UNIX Kommandos

auf der Grundlage des gleichnamen Skriptes[1]
von Tobias Oetiker (`oetiker@ee.ethz.ch`) und Thomas Gabathuler
erstellt von Michael Hoffmann (`hoffmann@inf.ethz.ch`)

24. Oktober 2005

1 Intro

Obwohl die meisten Rechner mit einer graphischen Benutzeroberfläche ausgerüstet sind, ist es immer noch das geschriebene Wort, welches einem am meisten Kontrolle über die Maschine gibt. Darum hier ein kurzer Abriss über einige der wichtigsten UNIX Kommandos. Eine Bemerkung gleich vorneweg: Gross– und Kleinschreibung sind unter UNIX signifikant!
Ein paar Anmerkungen zur Notation:

`Kommandos` und Dateinamen sind in Schreibmaschinenschrift gesetzt. Oftmals ist auch das Systemprompt (`username@slab5:~>`) mit angegeben.

<*Argumente*> sind Platzhalter und müssen im konkreten Fall jeweils durch die gewünschte Datei, das gewünschte Verzeichnis oder ähliches ersetzt werden.

`[Tasten]` bezeichnen entsprechende Tasten auf der Tastatur, durch deren Drücken sich bestimmte Effekte erzielen lassen. So steht beispielsweise `[CTRL]–[c]` für folgende Tastenkombination: `[CTRL]` (links auf der Tastatur) gedrückt halten und gleichzeitig einmal die Taste `[c]` drücken. `[SPACE]` ist übrigens die grosse lange Taste ganz unten. `:-)`

2 Die `tcsh`-Shell

Nach dem Aufstarten des Accounts erscheint auf dem Bildschirm ein weisses Fenster (`xterm`), in welches man UNIX Befehle eintippen kann. Das Programm welches in dem Fenster drin die Befehle entgegennimmt und verarbeitet heisst `tcsh` (sprich Ti-Si-Schell).

2.1 Ein paar Tips am Rande

1. Nach Drücken von `[TAB]` (`[`→|`]`) versucht die Shell, die Eingabe zu vervollständigen. Hierzu ein Beispiel:

   Die Eingabe von `cd LangerDirectoryName` kann abgekürzt werden mit `cd Lang` + Tabulatortaste. Allerdings nur, falls `Lang` eindeutig zuzuordnen ist. Das heisst, es existiert kein weiteres Directory, das mit `Lang` beginnt. Andernfalls wird eine Liste aller möglichen passenden Verzeichnisnamen angezeigt.

2. Mit den Pfeiltasten `[↑]` und `[↓]` kann man die letzten Eingaben auf die Eingabezeile zurückholen.

3. Falls mal etwas daneben geht: mit `[CTRL]–[c]` kann man die meisten Programme abbrechen.

3 Das Filesystem

```
username@slab5:~> ls
```

gibt eine Liste der Directoryeinträge im aktuellen Verzeichnis aus, die mit dem Parameter näher bestimmt werden können.
Verschiedene nützliche Optionen sind zum Beispiel:

[1] `http://www.ee.ethz.ch/isg/tardis/commands/`

ls -l	zeigt ausführlichere Informationen, insbesondere die File-Attribute (siehe Abschnitt 4).
ls -a	Einträge, deren Name mit einem “.” beginnen werden auch aufgelistet.
ls -R	Der ganze Sub-Baum wird auch angezeigt.
ls -F	Bei jeder Datei wird gleich mitangezeigt ob es sich dabei um ein Programm oder ein Verzeichnis handelt.

Natürlich können die Optionen kombiniert werden (Gross-/Kleinschreibung!).

```
username@slab5:~> cd <directoryname>
```

Damit wechselt man ins Verzeichnis <*directoryname*>. Anstelle von <*directoryname*> kann man auch .. schreiben, um im Verzeichnisbaum eine Stufe nach oben zu wechseln. Eine weitere nützliche Variante ist cd ~, womit man wieder im eigenen Home-Directory landet. Am Anfang jeder Eingabezeile wird jeweils der Name des aktuellen Verzeichnisses angezeigt.

```
username@slab5:~> mkdir <directory-namen>
```

Erzeugt ein neues Directory. Und zwar in dem Verzeichnis, in dem man sich gerade befindet.
Tja und so geht es weiter. Hier eine kleine Tabelle mit Befehlen und ihren Auswirkungen.

mv <*quelle*> <*ziel*>	um eine Datei umzubenennen.
mv <*quelle*> <*ziel-dir*>	um eine Datei in einem anderen Directory zu plazieren.
mv <*quellen-dir*> <*ziel-dir*>	um ein Directory umzubenennen oder es in einem anderen Directory zu plazieren.
cp <*quelle*> <*ziel*>	kopiert eine Datei.
cp <*quelle*> <*ziel-dir*>	kopiert eine Datei in ein anderes Directory unter Beibehaltung des Namens.
rm <*filename*>	löscht eine Datei. (Achtung, die Datei ist dann i.a. nicht wiederherstellbar!)
rmdir <*directoryname*>	löscht ein (leeres) Directory.
rm -r <*directoryname*>	löscht ein Directory inklusive Inhalt. (**Vorsicht!!!**)

Achtung DOS User: Sachen wie cp *.dat *.txt funktionieren nicht! Dazu später mehr.

4 Fileattribute

Jede Datei und jedes Directory, besitzt unter UNIX nicht nur einen Namen, sondern noch eine ganze Reihe weiterer Informationen. So ist zum Beispiel für jede Datei festgelegt, wem sie gehört und unter welchen Umständen sie gelesen und verändert werden darf.
Vor allem das mit den Lese und Schreibrechten ist eine interessante Sache, schauen wir uns doch einmal eine Datei von nahem an:

```
username@slab5:~ > ls -al
[...]
drwxr-xr-x  3 ifm2     ifm          512 Jul 12 17:03 ./
-rw-------  1 ifm2     ifm         1621 Jul 12 16:22 mbox
0123456789
```

10 Attribute (0-9) werden angezeigt.
Das *Attribut 0* gibt Auskunft über die *Art* des Eintrags: “-” steht für ein normales File. “d” zeigt an, dass es sich um ein Directory handelt.
Die *Attribute 1-9* regeln die *Zugriffsrechte* auf das File bzw. Directory und haben folgende Bedeutung: “r” lesbar, “w” schreibbar (also auch löschbar), “x” ausführbar (Bei einer normalen Datei bedeutet das,

dass es sich dabei um ein Programm handelt. Bei einem Directory, bedeutet das "`x`", dass man mit `cd` in das Verzeichnis hinein wechseln kann).
Die *Attribute 1-3* sind die sogenannten *user*-Attribute und regeln somit die Zugriffsrechte des Besitzers auf ein File bzw. Directory. Die *Attribute 4-6*, die *group*-Attribute, regeln analog die Zugriffsrechte einer *Gruppe*, in diesem Fall der Gruppe `ifm`, die nur aus dem Kursleiter-Account `ifm0` besteht. Die *Attribute 7-9*, die *others*-Attribute, regeln die Zugriffsrechte von jedem beliebigen Aussenstehenden.

5 Files oder Directories vor fremdem Zugriff schützen

Wenn immer du in Deinem Account ein File oder ein Directory erzeugst werden die Attribute automatisch wie folgt gesetzt:

Directory: `drwxr-xr-x`

File: `drw-r--r--`

Das bedeutet, dass **jedermann deine Directories einsehen, deine Files lesen und Programme ausführen kann. Auch neu erzeugte Files sind der Öffentlichkeit zur Einsicht freigegeben.** Dies entspricht der offenen Philosophie von UNIX, sollte jedoch nie vergessen werden!
Eine Ausnahme bilden "persönliche" Dateien wie mail– oder news-files. Diese werden automatisch mit Attributen erzeugt, die Einlesen durch andere verhindern.
Natürlich ist es auch möglich, die File-Attribute zu verändern. Hierzu dient das Kommando `chmod`, z.B. mit

```
username@slab5:~> chmod o-w <filename>
```

entzieht man *others* die Schreibrechte für `<filename>`. Ersetzt man `o` durch `u` bzw. `g`, ändert man die *user*– bzw. *group*-Rechte. Ersetzt man `-` durch `+`, werden die Rechte erteilt statt entzogen. Schliesslich kann man durch `r` und `x` statt `w` die Lese– bzw. Ausführbarkeitsrechte verändern. Noch ein Beispiel:

```
username@slab5:~> chmod g+r <filename>
```

macht `<filename>` für alle Mitglieder seiner Gruppe lesbar.

6 Wildcards

Die Zeichen "`*`", "`?`" und "`[xyz]`" (diesmal sind nicht die Tasten gemeint, sondern tatsächlich diese Zeichenfolge) werden von der `tcsh` speziell behandelt. Sie heissen *wildcards* und funktioniert folgendermassen:
Wenn immer die shell z.B. einen "`*`" in der Eingabezeile findet, wird dieser durch eine Liste der Dateinamen im aktuellen Verzeichnis ersetzt. Erst dann startet die shell das entsprechende Programm.
Wenn du also in einem Verzeichnis mit den Dateien `hallo.txt` und `timelord.txt` das command `ls *` eingibst, wird von der shell erstmal der "`*`" durch "`hallo.txt timelord.txt`" ersetzt und dann "`ls hallo.txt timelord.txt`" ausgeführt. Daher wird `ls` dann die Verzeichniseinträge für `hallo.txt` und `timelord.txt` zeigen.

6.1 Was bedeuten die Wildcards im einzelnen

`*`	Steht für beliebig viele (auch Null) Zeichen.
`?`	Ist Platzhalter für genau ein einzelnes Zeichen.
`[xyz]`	Steht für jedes in der Klammer aufgeführte Zeichen. Hier also x,y und z.
`[g-t]`	Ist Platzhalter für alle im Bereich enthaltenen Zeichen. Ein Bereich besteht aus zwei, durch ein "-" getrennte Zeichen. Hier also g bis t.

6.2 Hier noch einige weitere Beispiele

`ls -a .*`	Listet alle Directoryeinträge auf, die mit einem Punkt beginnen.
`ls -a [a-z]*`	Listet alle Directoryeinträge auf, die mit einem kleinen Buchstaben beginnen.
`ls *.*`	**(Achtung, DOS User!)** zeigt nicht wie gewohnt alle vorhandenen Files, sondern nur die, die einen Punkt im Namen enthalten.

7 emacs

Dieses Programm ist eigentlich schon eine Legende! "`emacs`" heisst "**e**diting **mac**ro**s**", wobei sich aber auch hartnäckige Gerüchte halten, der Name stehe eher für "eight megabytes and constantly swapping", eine Anspielung auf seinen enormen Speicherhunger. Er ist ein Texteditor, aber kein gewöhnlicher ... Es gibt eigentlich nichts, was er nicht kann. Die hohe Verfügbarkeit und seine speziellen Dienstleistungen, die er Programmierern bietet, machen ihn zu einem unschlagbaren Werkzeug.
Entgegen diversen Gerüchten, ist Emacs nicht besonders schwierig zu bedienen. Er verfügt über Pulldown Menüs und ein Online-Helpsystem. Obwohl fast alle Funktionen über Maus-Menus zugänglich sind, bevorzugen manche die Tastatur. Falls Ihr auch dazugehört: Auf Seite 8 findet Ihr eine Tabelle mit einigen häufig benötigten Tasten(-kombinationen). Für Leute mit viel Speicher und Icon-Sucht ist `xemacs` zu empfehlen – eine emacs Variante, welche in einigen Bereichen massiv erweitert worden ist.

8 Fileinhalt anzeigen mit `less`

```
username@slab5:~> less <filename>
```

Less zeigt eine Datei seitenweise an. Mit den Pfeiltasten kann man sich darin hin und her bewegen. `[?]` zeigt einem alle verfügbaren Kommandos an, mit `[/]` kann man die Datei durchsuchen und `[q]` beendet das Programm.

9 Drucken

Irgendwann seid ihr sicherlich nicht mehr damit zufrieden, alle eure Texte nur am Bildschirm zu betrachten, sondern möchtet etwa eure Semesterarbeit zwecks Abgabe auf Papier ausgedruckt haben. Klar, dass das unter UNIX auch geht, schliesslich würden sonst die netten Laserdrucker völlig unnötig rumstehen. Generell sei aber gesagt, dass echte UNIX-Wizards ihren Output auf Papier auf das absolut Notwendige reduzieren; UNIX bietet nicht umsonst so viele Möglichkeiten, die eigene Arbeit am Bildschirm zu bewundern.

9.1 Tips zum Drucken

Kontrolliert Ausdrucke zuerst am Bildschirm, das erspart viel ärger. Bitte startet keine Druckjobs beim Ausloggen, ich weiss, dass das geht und toll ist, aber es ist ausserordentlich unfair gegenüber euren MitstudentenInnen, falls etwas schiefgeht (denkt an Murphy's law)! Wer zu Stosszeiten mehrere MByte ausdruckt, ist für seine Sicherheit selbst verantwortlich.

9.2 Wie drucke ich etwas aus?

Mit VPP[2] kannst Du von allen Rechnern aus drucken. Der Ausdruck kommt in einer VPP-Station aus dem Drucker. Um den Sourcecode eines Programms `hello.c` auszudrucken kannst Du in einer Shell

```
username@slab5:~> vpp -dest=HGE22 -form=listq.ds hello.c
```

eingeben. Damit kommt Dein Programm zweispaltig, doppelseitig bedruckt aus dem Drucker.

[2] `http://www.sd.id.ethz.ch/vpp/vpp.html`

Benutze bitte für **Textdateien** die Optionen

`-form=listq.ds`	(2 Spalten Querformat, doppelseitig)
`-form=listh.ds`	(2 Spalten Hochformat, doppelseitig)

und bei **Postscript-Dateien** das Programm `psnup`:

```
username@slab5:~> psnup -2 in.ps out.ps
username@slab5:~> vpp -dest=HGG26 -dev=pts -form=a4.ds out.ps
```

10 Prozesse

Unter UNIX laufen immer eine ganze Menge von Programmen (Prozessen) gleichzeitig. Mit den Befehlen `ps` und `top` kann man sich die Sache ansehen. Besonders interessant ist top, da es die Prozesse gleich noch sortiert nach Belastung des Computers.
Ein Kurze Befehlsübersicht zu `top`:

u <*username*>	Zeigt nur die Prozesse eines bestimmten Benutzers.
k <*PID*>	Fordert ein Programm auf, sich zu beenden.
k -9 <*PID*>	Killt ein Programm. Wer nicht hören will, ...

10.1 Programme im Hintergrund laufen lassen.

Hat man ein Programm aufgerufen, so bleibt die Shell "hängen", bis dieses beendet wird. Das kann verhindert werden, indem man das Programm im Hintergrund laufen lässt. Dies wird erreicht, indem man dem Programmnamen und den Parametern ein "`&`" anhängt.
Alternativ lassen sich Programme auch mit [CTRL]–[z] und dem Befehl `bg` in den Hintergrund befördern.

11 Absturz

Auch unter UNIX kann das einmal vorkommen. Um das Problem zu beheben, loggt man sich an einem Nachbarrechner ein, übernimmt von dort aus mit `ssh` die Kontrolle der abgestürzten Maschine und schiesst das abgestürzte Programm mit `kill -9` ab. **Achtung:** die Computer nie abschalten. Erstens kann dadurch die Harddisk beschädigt werden, und zweitens ist es bei UNIX nur sehr selten so, dass sich das ganze System aufhängt. Oft arbeiten andere Leute "remote" auf der Maschine, ohne Probleme zu haben.

12 Output umleiten

```
username@slab5:~> <command> > <filename>
```

schreibt die Daten, die der Befehl liefert, in ein File.

```
username@slab5:~> ls -l > savedir.txt
```

erzeugt ein File namens savedir.txt, das eine Liste der Dateien im aktuellen Verzeichnis enthält

```
username@slab5:~> <command> >> <filename>
```

fügt die Daten, die der Befehl liefert, hinten an einen File an.

13 Pipe

```
username@slab5:~> <command1> | <command2>
```

leitet die Ausgabe des ersten Befehls direkt zum zweiten Befehl, der die kommenden Daten als Eingabe weiterverarbeitet.

```
username@slab5:~> ls -l | less
```

gibt Directoryeinträge seitenweise aus.
Unter UNIX existieren sehr viele Programme, die speziell für die Verwendung mit Pipes gemacht sind, die sogenannten *Filter*. Hier eine kleine Auswahl:

`wc`	zählt Wörter und Linien.
`head -`*x*	zeigt die ersten x Zeilen.
`tail -`*x*	zeigt die letzten x Zeilen.
`grep` *xyz*	zeigt die Zeilen, die *xyz* enthalten.
`sort`	sortiert die Zeilen.

Natürlich lassen sich diese Filter auch ohne Pipe verwenden: `grep main *.c` sucht das Wort “`main`” in allen Dateien, die auf `.c` enden. Pipes lassen sich übrigens auch verketten: `ls * | sort | less`.

14 Environmentvariablen

Gewisse Informationen werden unter UNIX in sogenannten Environmentvariablen abgelegt. Das funktioniert folgendermassen: Wann immer ein Programm aufgerufen wird, erhält es eine Kopie aller Environmentvariablen seiner Eltern mit auf den Weg. Auf diese Weise können Informationen weitergegeben werden, ohne dass sie in irgend einer Datei gespeichert werden müssen. Mit folgenden Befehlen kannst Du den Inhalt von Environmentvariablen ansehen und verändern:

`setenv XYZ "hi world"`	Weist der Variable `XYZ` den Wert “`hi world`” zu.
`unsetenv XYZ`	Löscht die Variable `XYZ`.
`echo $XYZ`	Zeigt den Inhalt von `XYZ`.
`printenv`	Zeigt **alle** Environmentvariablen, die zur Zeit definiert sind.

Hier eine Aufstellung einiger wichtiger Environmentvariablen:

`EDITOR`	Der Name des Programms, das als Editor verwendet werden soll.
`PAGER`	Der Name des Programms, das zum Anzeigen von Textdateien verwendet werden soll.
`PATH`	Eine durch “`:`” getrennte Liste der Verzeichnisse in denen nach Programmen gesucht wird.

14.1 Aliases

Eine sehr praktische Funktion der `(t)csh` sind die aliases. Damit können auf einfache Art und Weise neue Befehle erzeugt werden.

`alias`	Zeigt eine Liste der momentan definierten aliases.
`alias` <*name*> <*text*>	Erzeugt ein neues alias (<*name*>), das den Befehl (<*text*>) ausführt.

Ist man es leid, `ls -al` zu tippen, und möchte dafür eine Abkürzung, zum Beispiel `la`, definieren, so geht man folgendermassen vor:

```
username@slab5:~> alias la "ls -al"
```

Diese Definition geht allerdings beim Ausschalten wieder verloren. Um das zu umgehen, kannst du die alias Definition in dein `~/.alias` file einfügen.

15 .tcshrc und .login

Wenn du dich in dein Konto einloggst, werden folgende Dateien automatisch ausgeführt, bevor du irgendwelche Eingriffsmöglichkeit bekommst:

1. `/etc/.cshrc`
2. `~/.tcshrc`
3. `/etc/.login`
4. `~/.login`

Die beiden Dateien im `/etc` Verzeichnis kannst du nicht verändern. Sie enthalten allgemeine Konfigurationsanweisungen, die für jedes Konto gelten. Die Dateien in deinem Home Verzeichnis (`~/.tcshrc` und `~/.login`) kannst du jedoch mit einem Texteditor verändern.
Während die `.login` Dateien nur beim login gelesen werden, werden die beiden `.(t)cshrc` jedesmal, wenn du eine neue `tcsh` startest, gelesen.
Bei Bedarf kann man die `.login` und `.tcshrc` Dateien auch mit dem Befehl `source .login` oder `source .tcshrc` gezielt ausführen.

16 Ich brauche Hilfe

Einer der wichtigsten Befehle unter UNIX dürfte wohl `man` (Abkürzung für Manual = Handbuch) sein. Dieser gibt Auskunft über Funktion, Anwendung und Optionen von Befehlen. Natürlich gibt `man` auch gleich Auskunft über sich selbst. Die Ausgabe erfolgt Seitenweise. Um die nächste Seite anzuzeigen, muss `[SPACE]` kurz angetippt werden, mit `[b]` lässt sich eine Seite zurückblättern. Hat man einmal die falsche Seite erwischt oder die gewünschte Information bereits gefunden, kann man **mit `[q]` die Ausgabe abbrechen.**

```
username@slab5:~> man man

MAN(1)                    USER COMMANDS              MAN(1)

NAME
   man - display reference manual pages; find reference
         pages by keyword

SYNOPSIS
   man [-] [-t] [-M path] [-T macro-package]  [[section]  title...] ...
   man [-M path] -k keyword ...
   man [-M path] -f filename ...

DESCRIPTION
   man displays information from (...)
```

Manpages enthalten viele Informationen in sehr konzentrierter Form. Daher erfordert das Lesen eine gewisse Übung. Selbst wenn man nicht mit allem, was man dort findet, sofort etwas anfangen kann, so erhält man selbst beim Überfliegen und durch gezieltes Herauspicken von Stichwörtern schnell wichtige Hinweise.
Oftmals kennt man jedoch gerade die Schreibweise eines Befehls nicht. Hier kommt der Befehl `apropos` zum Zug. Damit lassen sich alle manpages finden, die ein bestimmtes Stichwort enthalten.

```
username@slab5:~> apropos shell
```

Bei Syntaxfehlern wird oft eine kurze Erklärung über den korrekten Gebrauch des Befehls ausgegeben. Meistens ist diese kleine Hilfe auch mit der Option `-h` oder `--help` zu erreichen.

```
username@slab5:~> mv --help
```

Wichtige Tasten für den `emacs`

Dateien:	
`[CTRL]-[x] [CTRL]-[c]`	`emacs` beenden.
`[CTRL]-[x] [CTRL]-[f]`	Datei laden.
`[CTRL]-[x] [CTRL]-[s]`	Datei speichern.
`[CTRL]-[x] [k]`	Datei schliessen.
Fenster:	
`[CTRL]-[x] [Ø]`	Fenster schliessen.
`[CTRL]-[x] [1]`	alle anderen Fenster schliessen.
`[CTRL]-[x] [2]`	Fenster horizontal teilen.
Cursorbewegung:	
`[←], [→], [↑], [↓]`	Cursor zeichenweise nach links, rechts, oben oder unten bewegen.
`[CTRL]-[←]`	Cursor wortweise vorwärts bewegen.
`[CTRL]-[→]`	Cursor wortweise rückwärts bewegen.
`[CTRL]-[↓]`	Cursor seitenweise vorwärts bewegen.
`[CTRL]-[↑]`	Cursor seitenweise rückwärts bewegen.
`[HOME]`	Cursor an den Anfang der Zeile bewegen.
`[END]`	Cursor an das Ende der Zeile bewegen.
`[CTRL]-[HOME]`	Cursor an den Anfang der Datei bewegen.
`[CTRL]-[END]`	Cursor an das Ende der Datei bewegen.
Löschen und Einfügen:	
`[CTRL]-[k]`	Lösche Rest der Zeile.
`[CTRL]-[w]`	Lösche markierten Bereich. (*Cut*)
`[CTRL]-[y]`	Füge zuletzt gelöschtes wieder ein. (*Paste*)
Suchen:	
`[CTRL]-[s]`	Zeichenkette suchen (vorwärts).
`[CTRL]-[r]`	Zeichenkette suchen (rückwärts).
`[ESC]-[%]`	Suchen und Ersetzen (vorwärts).
Sonstiges:	
`[CTRL]-[g]`	Aktuelle Funktion beenden. (= `[CTRL]-[c]` in der shell)
`[CTRL]-[_]`	Macht letzte Änderung rückgängig. (*Undo*)
`[CTRL]-[h] [k]`	Beschreibt die Funktion der anschliessend gedrückten Taste.
Im `C++`-mode:	
`[f6]`	Alle Zeilen der Datei korrekt einrücken.
`[f7]`	Zum vorhergehenden Fehler, den der Compiler gefunden hat.
`[f8]`	Zum nächsten Fehler, den der Compiler gefunden hat.
`[f9]`	Datei compilieren (mit `make`).
`[f10]`	Geh zu Zeile # …