

# Kurzanleitung zur UNIX Shell

Institut für Mikroelektronik, 8. Oktober 2009

## 1 GNU/Linux Shell

Die Shell unter GNU/Linux ist mit einem Eingabeaufforderungs- bzw. DOS-Fenster unter Windows vergleichbar. Es gibt mehrere Shells unter GNU/Linux, die man verwenden kann. Diese Einführung wird sich mit der **bash** (Bourne-again shell) beschäftigen.

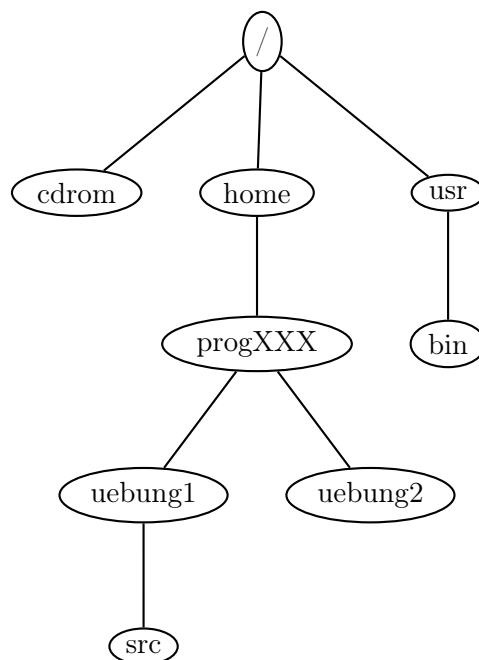
Um zur Shell zu gelangen, ist ein Terminal-Fenster — in den Übungen **terminal**, in manchen Linux Umgebungen auch **konsole** oder **gnome-terminal** — zu öffnen.

## 2 Dateisystem

### 2.1 Allgemein

Im Vergleich zu Windows ist das Dateisystem anders aufgebaut. Es gibt keine Laufwerksbuchstaben sondern nur das **root**-Verzeichnis (/) und Unterordner. Verzeichnisse werden im Gegensatz zu Windows nicht mit \ (Backslash) sondern mit / (Slash) getrennt. Festplatten,

Abbildung 1: Dateibaum



das CDROM-Laufwerk, etc. können unter jedem beliebigen Unterordner eingehängt (**mount**) werden. Wird z.B. das CDROM-Laufwerk unter /cdrom eingehängt, so wird der Inhalt der CD im Ordner /cdrom angezeigt.

In den folgenden Beispielen wird der Dateibaum aus Abbildung 1 verwendet.

## 2.2 Dateinavigation

Der Name des aktuellen Ordners kann mit dem Befehl **pwd** (print working directory) angezeigt werden. Um zu einem anderen Ordner zu navigieren, muss man (wie unter DOS) den Befehl **cd** (change directory) verwenden. Um in das Home-Verzeichnis zu gelangen, muss man **cd** ohne Argument ausführen. Der Befehl **ls** (list segments) zeigt den Inhalt eines Ordners an. Wird kein Argument übergeben, so wird der Inhalt des aktuellen Ordners angezeigt. Alle Ordnerangaben können relativ vom aktuellen Verzeichnis aus oder absolut mit beginnendem / angegeben werden.

Beispiele

```
1 progXXX@lsXXX ~ $ pwd
2 /home/progXXX/
3 progXXX@lsXXX ~ $ ls
4 uebung1 uebung2
5 progXXX@lsXXX ~ $ cd uebung1/src
6 progXXX@lsXXX ~/uebung1/src $ cd
7 progXXX@lsXXX ~ $ cd /cdrom
8 progXXX@lsXXX /cdrom $ cd ..
9 progXXX@lsXXX / $
```

**progXXX@lsXXX ~ \$** ist der Prompt, den die Shell ausgibt. **progXXX** steht dabei für den aktuellen Usernamen, **lsXXX** für den Rechnernamen und **~** für das aktuelle Arbeitsverzeichnis.

Tabelle 1: Pfadangaben

.	Das aktuelle Verzeichnis
..	Das übergeordnete Verzeichnis
~	Das Heimverzeichnis (in unseren Beispielen /home/progXXX)
/home/progXXX/uebung1	Eine absolute Pfadangabe
uebung1/src	Eine relative Pfadangabe

## 2.3 Dateisystemoperationen

Ordner können mit dem Befehl **mkdir** (make directory) angelegt werden. Analog zu **mkdir** kann man mit **rmdir** (remove directory) einen Ordner löschen. Der Ordnerinhalt muss dazu leer sein! Dateien können mit dem Befehl **rm** (remove) gelöscht werden. **rm** unterstützt weiters die Option **-r** mit der das Verzeichnis und sämtliche Unterverzeichnisse gelöscht werden. Ähnlich wie bei DOS werden die *Wildcards* \* (steht für eine beliebige Zeichenfolge) und ? (steht für ein beliebiges Zeichen) unterstützt.

**Achtung!** Es gibt keinen Papierkorb! Dateien, die gelöscht wurden, sind nicht wiederherstellbar. Also immer darauf achten, dass die richtigen Dateinamen angegeben werden.

Linux fragt beim Löschen von Dateien nicht nach, ob diese wirklich gelöscht werden sollen. Hierbei ist besonders mit der **-r** Option Vorsicht geboten. Ein Tippfehler kann fatale Folgen haben.







Dateien und Ordner können mit dem Befehl **mv** (move) verschoben beziehungsweise umbenannt werden.

## Beispiele

```
1 progXXX@lsXXX ~ $ mkdir uebung3
2 progXXX@lsXXX ~ $ cd uebung3
3 progXXX@lsXXX ~/uebung3 $ mkdir src
4 progXXX@lsXXX ~/uebung3 $ cd ..
5 progXXX@lsXXX ~ $ mkdir uebung3/tmp
6 progXXX@lsXXX ~ $ ls
7 uebung1 uebung2 uebung3
8 progXXX@lsXXX ~ $ ls uebung3
9 src tmp
10 progXXX@lsXXX ~ $ mv uebung3 old
11 progXXX@lsXXX ~ $ ls
12 old uebung1 uebung2
13 progXXX@lsXXX ~ $ rm old
14 rm: cannot remove 'old': Is a directory
15 progXXX@lsXXX ~ $ rmdir old
16 rmdir: failed to remove 'old': Directory not empty
17 progXXX@lsXXX ~ $ rm -r old
18 progXXX@lsXXX ~ $ ls
19 uebung1 uebung2
```

### 2.4 Dateien betrachten

Um Dateien in der Shell betrachten zu können, gibt es zwei wichtige Befehle: **cat** (concatenate) und **less**. **cat** gibt die ganze Datei auf dem Bildschirm aus. Hingegen zeigt **less** nur so viel an, wie in den Bildschirm passt und man kann mit den Cursortasten scrollen. Mit **less** kann man auch in einer Datei suchen. Dazu  gefolgt vom Schlagwort und  eingeben. Sind vom Schlagwort weitere Ergebnisse vorhanden,  (next) drücken um zum nächsten Treffer zu gelangen. Das Programm kann mit der Taste  verlassen werden.

### 2.5 bash-Hilfen

Die letzten Befehle können mit Hilfe von den Cursortasten nochmals aufgerufen werden. Weiters ist die Shell in der Lage Befehle und Dateien mittels Tab-Taste zu vervollständigen. Drückt man die Tab-Taste zwei mal schnell hintereinander, so werden alle Möglichkeiten aufgelistet, vorausgesetzt es gibt mehrere.

## 3 Befehlshilfe

Ein weiterer wichtiger Befehl ist: **man** (manual). Zur Betrachtung der Hilfedateien **man** eingeben. Es existieren auch Hilfedateien zu C-Header und zu C-Funktionen. Die Hilfeseiten sind in mehrere Kategorien eingeordnet, die mit Zahlen bezeichnet sind. 1 für die Befehle und 3 für die C-Funktionen. Die Kategorie muss man nur dann angeben, wenn ein und derselbe Name in mehreren Kategorien existiert (zum Beispiel *printf*).


## Beispiele

```
1 progXXX@lsXXX ~ $ man 3 printf
2 progXXX@lsXXX ~ $ man strcpy
3 progXXX@lsXXX ~ $ man string
```

Zeile 3 zeigt die Hilfeseite zur Includedatei string.h an. **man** stellt die Hilfeseiten mit Hilfe von **less** dar.

Die Hilfeseiten können und sollen während allen Übungen verwendet werden. Sie enthalten eine detaillierte Beschreibung der Funktionen, des Funktionsaufrufs mit genauer Parameterdefinition, in welchen Header Dateien sie zu finden sind und Erklärungen zum Rückgabewert.

## 4 Ausführen von Programmen

Programme kann man mit der Eingabe des Befehls gefolgt von der -Taste ausführen. In den vorherigen Beispielen wurde das schon öfter durchgeführt (**man**, **ls**, **cd**, ... sind alles Programme). Der Text, der nach dem Programm angegeben wird, sind die Argumente. Um Optionen von anderen Programmparametern leichter unterscheiden zu können, werden Optionen mit einem einzelnen - bzw. bei mehrbuchstabigen Optionen mit -- angegeben. Zum Beispiel `rm -r test` beziehungsweise `rm --recursive test`.

Es können nur Programme auf diese Weise ausgeführt werden, die in dafür konfigurierten Verzeichnissen liegen. Will man Programme ausführen, die im aktuellen Verzeichnis bzw. Unterverzeichnissen liegen, so muss man den Pfad dazu angeben:

```
Beispiele
1 progXXX@lsXXX uebung1/src $ ./uebung1
2 progXXX@lsXXX ~ $ uebung1/src/uebung1
```

Im Beispiel ist `uebung1` im Verzeichnis `uebung1/src` eine ausführbare Datei. Mit `./` kann man das aktuelle Verzeichnis ansprechen.

## 5 Kompilieren von Programmen mit gcc

Mit Hilfe des Programmes **gcc** (GNU Compiler Collection bzw. früher GNU Compiler) können C-Programme in ausführbaren Maschinencode übersetzt werden.

```
Beispiele
1 progXXX@lsXXX ~ $ gcc uebung1.c
2 progXXX@lsXXX ~ $ gcc -o test uebung1.c
3 progXXX@lsXXX ~ $ gcc -c uebung1.c
4 progXXX@lsXXX ~ $ gcc -o test uebung1.o
```

Die Erzeugung eines ausführbaren Programms kann in den eigentlichen Compilervorgang, bei dem C-Code in eine Objektdatei übersetzt wird, und das Linken, bei dem (auch mehrere) Objektdateien zu einem Executable zusammengeführt werden, unterteilt werden. Standardmäßig wird ein ausführbares Programm `a.out` erzeugt (Anweisung 1). Mit der Option `-o` kann man den Namen der Ausgabedatei angeben. Durch Angabe der Option `-c` wird nur der eigentliche Compilervorgang ausgeführt.

Anweisung 2 erzeugt eine ausführbare Datei mit dem Namen `test`, Anweisung 3 eine Objektdatei mit dem Namen `uebung1.o` und Anweisung 4 erzeugt aus der Objektdatei `uebung1.o` die ausführbare Datei `test`. Weitere Informationen befinden sich im Kapitel 3 des Buches.

## 6 Texteditor

Linux verfügt über eine Reihe von Editoren, von welchen sich einige nur durch kryptische Tastenkombinationen bedienen lassen. Auf den Übungsrechnern ist der Editor **gedit** vorhanden, der sich aber sehr einfach bedienen lässt und auch die C-Syntax geeignet darstellt. Durch Eingabe von **gedit prog1.c** wird die Datei `prog1.c` zum Editieren geöffnet. Auf anderen Linux Derivaten gibt es zum Beispiel **kate**.

## 7 make

Damit der Compiler nicht jedesmal von Hand aufgerufen werden muss, gibt es das Programm **make**.

**make** kann auch Abhängigkeiten auflösen, führt also alle Befehle und Regeln in der richtigen Reihenfolge aus und auch nur die Regeln, die Dateien betreffen, die sich seit dem letzten Aufruf von **make** verändert haben.

### 7.1 Makefile

Um **make** verwenden zu können, muss man im betroffenen Verzeichnis die Datei **Makefile** anlegen. Diese muss folgenden Inhalt besitzen:

```
Makefile
1 Regelname1: Abhaengigkeit1 Abhaengigkeit2
2     ↪Aktion1
3     ↪Aktion2
4
5 Regelname2: Abhaengigkeit3
6     ↪Aktion3
```

Achtung! Die Aktion muss mit einem Tabulator (  ) eingerückt sein.



Ein **Makefile** kann beliebig viele Regeln enthalten. Abhängigkeiten können Regeln und Dateien sein. Mehrere Abhängigkeiten müssen dabei mit Leerzeichen getrennt sein. Die Aktionen (eine Liste von Befehlen) werden von **make** beim Ausführen einer Regel abgearbeitet.

Folgendes **Makefile** erzeugt aus der Datei **hello.c** das ausführbare Programm **hello\_world**.

```
Makefile
1 all: hello.c
2     ↪gcc -o hello_world hello.c
```

Zeile 1 erstellt eine Regel mit dem Namen **all**, die von der Datei **hello.c** abhängt. Danach wird die Aktion in Regel **all**, also der Befehl

**gcc -o hello\_world hello.c**, ausgeführt.

An **make** kann die Regel übergeben werden, die ausgeführt werden soll. Wird keine Regel übergeben, so führt **make** die erste in der Datei aus.

Das vollständige **Makefile** inklusive einer Regel für die Generierung der Objekdatei **hello.o** würde so aussehen:

```
Makefile
1 all: hello.o
2     ↪gcc -o hello_world hello.o
3 hello.o: hello.c
4     ↪gcc -c -o hello.o hello.c
```

```
Beispielaufruf
1 progXXX@lsXXX ~ $ make
2 gcc      -c -o hello.o hello.c
3 gcc -o hello_world hello.o
4 progXXX@lsXXX ~ $ ls
5 Makefile hello.c hello_world
```

**make** besitzt darüber hinaus eine Reihe vordefinierter, automatischer Regeln. Folgendes Makefile generiert ebenfalls aus der Datei `hello.c` das ausführbare Programm `hello_world`. Mit dieser hierbei verwendeten vordefinierte Regel wird automatisch aus C-Dateien die jeweils dazugehörige Objektdatei erzeugt.

```
----- Makefile -----  
1 all: hello.o  
2     ↗gcc -o hello_world hello.o
```

Zeile 1 erstellt eine Regel mit dem Namen `all`, die von der Datei `hello.o` abhängt. Da die Datei `hello.o` nicht vorhanden ist, verwendet **make** die bereits vordefinierte Regel, um aus `hello.c` die Datei `hello.o` zu erzeugen. Danach wird die Aktion in Regel `all`, also der Befehl `gcc -o hello_world hello.o`, ausgeführt.

Ein weiteres Beispiel:

```
----- Makefile -----  
1 all: part1 part2  
2 part1: hello.o  
3     ↗gcc -o hello_world hello.o  
4 part2: bsp2.o  
5     ↗gcc -o rechner bsp2.o
```

In diesem Beispiel können mit `make part1` bzw. `make part2` die zwei Programme kompiliert werden. Durch die Eingabe von `make all` oder `make` werden automatisch beide Programme erstellt.

## 8 Cheatsheet

### 8.1 Verzeichnisoperationen

Befehle:

`ls` Verzeichnisinhalt anzeigen  
`cd` Verzeichnis wechseln  
`cd -` In vorheriges Verzeichnis wechseln  
`pwd` Aktuellen Pfad anzeigen  
`mkdir` Verzeichnis erstellen  
`rmdir` Verzeichnis löschen

Spezielle Pfade:

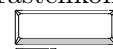





`.` Aktuelles Verzeichnis  
`..` Übergeordnete Verzeichnis  
`~` Heimverzeichnis

### 8.2 Dateioperationen

`rm` Datei löschen  
`rm -r` Ordnerstruktur löschen  
`cp` Datei kopieren  
`cp -r` Ordnerstruktur kopieren  
`mv` Datei/Ordner umbenennen/verschieben  
`cat` Datei betrachten  
`less` Datei betrachten (siehe less)







### 8.3 less

Tastenkombinationen in `less`:

	Einen Bildschirm nach unten scrollen
	Eine Zeile nach unten scrollen
	Suchen (Richtung Dateieende)
	Zum nächsten Vorkommen springen
	Suchen (Richtung Dateianfang)
	Zum vorherigen Vorkommen springen

Es können auch die Pfeiltasten verwendet werden.

### 8.4 Shellbefehle

	Vorheriger Befehl
	Befehl/Datei vervollständigen
 	Vervollständigungsmöglichkeiten anzeigen
 	In den vergangenen Befehlen suchen

### 8.5 Hilfe

`man thema` bzw.

`man 3 thema` für C-Funktionen.

### 8.6 gcc

`gcc -o ausgabedatei datei1.c datei2.c`

Optionen:

`-o datei` Ausgabedatei  
`-c` Nicht linken