Arbeiten unter Linux

In unseren Lehrveranstaltungen arbeiten wir mit der Kommandozeile, da wir nur so wirklich wissen, was wir eigentlich tun. Diese Kommandoeingabe ist die Basis für das Ausführen von Kommandos und wird vor allem unter Windows-Nutzern auch Konsoleneingabe genannt. Der Rest dieser kleinen Anleitung fasst die wichtigsten Kommandos zusammen. Unter Linux nennt man diese Tools auch Terminal, X-Term(-inal) oder Pseudoterminal, da ein echtes Terminal lediglich „simuliert“ wird.

Öffnen der Konsole:

Hierfür haben wir die folgenden Möglichkeiten:

1. Drücken der beiden Tasten `Alt`+`F2`, eingeben des Befehls `gnome-terminal` und bestätigen mit der `Enter`-Taste. Statt `gnome-terminal` kann man auch `xterm` oder `x-terminal-emulator` eingeben.
2. Aktivieren des Menüeintrages `Anwendungen` → `Zubehör` → `Terminal`

Starten und Beenden der Shell:

In so einem Terminal läuft ein Programm, das Shell oder Kommandointerpreter genannt wird. Diese Shell liest die Tastatureingaben, interpretiert diese als Kommandos und startet die entsprechenden Programme. Anschließend wartet die Shell so lange, bis das Programm zu Ende ist, um anschließend die nächste Kommandoeingabe zu lesen und zu bearbeiten. Falls einem die automatisch gestartete Shell nicht gefällt, kann man durch Eingabe des Kommandos `bash` eine sehr komfortable Shell starten und diese mittels `exit` wieder beenden.

Navigieren innerhalb des File-Systems:

Jede Shell verwaltet ein *aktuelles Verzeichnis*, das auch als *working directory* bezeichnis wird. Das aktuelle Verzeichnis kann mittels des Kommandos `cd [verzeichnis]` verändert werden. Hierbei wird das Argument `verzeichnis` wie folgt verstanden:

1. Relativ zum Heimatverzeichnis, wenn das Argument mit einer Tilde ∼ anfängt.
2. Relativ zur Wurzel des File-Systems (auch absolut genannt), wenn das Argument mit einem `/` anfängt.
3. Relativ zum aktuellen Verzeichnis, wenn obige Bedingungen nicht erfüllt sind.

Die folgenden Beispiele dienen der Illustration:

Befehl	Wirkung
`cd ex1`	Wechsel in das Verzeichnis `ex1`
`cd ..`	Wechsel in das Oberverzeichnis (parent directory)
`cd /tmp`	Wechsel durch Angabe eines absoluten Pfadnamens
`cd`	Wechsel in das Heimatverzeichnis
`cd ∼`	Wechsel in das Heimatverzeichnis
`cd ∼/ex2`	Wechsel `ex2` unterhalb des Heimatverzeichnisses

Darüber hinaus sind noch folgende Kommandos wichtig:

Befehl	Wirkung
`mkdir xyz`	Einrichten eines neuen Verzeichnises `xyz`
`rmdir xyz`	Löschen des Verzeichnises `xyz`, sofern es leer ist
`rm datei`	Löschen der Datei `datei`
`more datei`	Seitenweises Anzeigen der Datei `datei`
`ls`	Auflistung aller Dateien im aktuellen Verzeichnis
`ls dir`	Auflistung aller Dateien im Verzeichnis `dir`
`mv old new`	Umbennen der Datei (des Verzeichnisses) `old` in `new`

Erstellen von Dateien: der Editor:
Ein einfacher Editor ist unter dem Namen `gedit` aufrufbar. Dieser Editor kann in einfacher Weise gemäß des persönlichen Geschmacks konfiguriert werden. Beim Aufruf des Editors kann auch direkt ein Dateiname angegeben werden. Beispiel: Anlegen einer Datei mit dem Namen `ueb1.c`: `gedit ueb1.c`

Der C-Compiler: Unter Linux wird üblicherweise der GNU C-Compiler `gcc` verwendet. `gcc` ist aber nicht nur *ein* Programm sondern besteht aus einem ganzen Familie einzelner Programme. Insbesondere erfordert jede Programmiersprache seinen eigenen Compiler. Die gute Nachricht: `gcc` kann die betreffende Programmiersprache aus dem Dateinamen ableiten. Hier ein paar Beispiele inklusive der Ausgabe von Warnungen:

Befehl	Sprache	ausführbare Datei
`g++ -Wall -o hallo hallo.cpp`	C++	`hallo`
`gcc -Wall -o hallo hallo.c`	C	`hallo`

Dokumentation:
Unter Unix-artigen Betriebssystemen ist in der Regel die Dokumentation alle Befehle und Standardfunktionen direkt verfügbar. Hierfür gibt es das Kommando `man`. Hierzu folgende Beispiele:

Befehl	Wirkung
`man ls`	Ausgabe der Dokumentation zum `ls`-Kommando
`man cd`	Ausgabe der Dokumentation zum `cd`-Kommando
`man man`	Beschreibung des `man` Kommandos
`man strcpy`	Beschreibung der C-Funktion `strcpy()`

Starten von Programmen im Hintergrund: Wie oben beschrieben wartet die Shell bis sich ein aufgerufenes Programm wieder beendet hat. Das schränkt die Möglichkeiten, insbesondere bei komplexen Programmen, sehr ein. Daher kann man Programme so starten, dass die Shell nicht auf deren Ende wartet. Dies nennt man „Programme im Hintergrund laufen lassen.“ Hierzu muss die Kommandozeile mit einem Undzeichen-`&` abgeschlossen werden. Beispiel: `gedit ueb2.cpp &`