Grundzüge von Unix

Überblick

Unix ist ein sehr flexibles Multiuser-Betriebssystem aus den 1970er Jahren, das im technisch-naturwissenschaftlichen Bereich weit verbreitet ist. Mit der Verfügbarkeit von "freien" Versionen wie NetBSD und Linux hat Unix aber auch in vielen anderen Bereichen, vor allem im Bildungssektor, Einzug gehalten. Unix war und ist an sich ein Kommandozeilen-orientiertes Betriebsystem, doch wurde Mitte der 1980er Jahre das Graphiksystem X-Windows entwickelt, auf dem zahlreiche Anwendungen aufbauen und von dem es ebenfalls eine freie Implementation gibt.

"Multiuser"-System heißt, daß mehrere Benutzer zur selben Zeit auf dem System arbeiten können. Zusätzlich kann jeder Benutzer auch noch mehrere Prozesse gleichzeitig laufen lassen. Die scheinbare Gleichzeitigkeit kommt dadurch zustande, daß die insgesamt zur Verfügung stehende Rechenleistung in kleinste zeitliche Portionen unterteilt und der Reihe nach bzw. nach Prioritäten gewichtet den einzelnen Prozessen zugeordnet wird.

Um in einer Multiuser-Umgebung überhaupt Zugang zu den Betriebsmitteln des Systems zu bekommen, müssen die Benutzer über eine Zugangsberechtigung (Account) verfügen und sich gegenüber dem System durch einen Namen und ein Passwort identifizieren. Die Datenbereiche der einzelnen Benutzer sind zunächst streng getrennt, doch kann man Benutzer in verschiedene Gruppen zusammenfassen und mit Hilfe von Zugriffsrechten regeln, zu welchen Files die Mitglieder einer Gruppe Zugang haben sollen.

Unix ist ein minimalistisches Betriebssystem in dem Sinn, daß es Systemprogramme nur für eher eng definierte Aufgaben gibt. Zur Lösung komplexerer Aufgaben kann man aber oft eine Reihe von einfacheren Programmen kombinieren.

Schließlich gibt es auch kein Unix "an sich", sondern neben den freien Implementationen noch eine Reihe von kommerziellen Versionen der verschiedensten Hersteller. Alle diese Varianten sind einander zwar sehr ähnlich, können sich aber in Details der Organisation des Filesystems, dem Vorhandensein und der genauen Syntax einzelner Kommandos usw. unterscheiden.

Filesystem

Das Filesystem von Unix ist, wie das der meisten Betriebssysteme, baumartig–hierarchisch in Verzeichnissen organisiert. An der Spitze des Baumes steht das Root-Verzeichnis

`/`

und unmittelbar darunter eine Reihe von Systemverzeichnissen wie

`/usr`, `/var`, `/dev`, ..., `/home`

Im letzteren befinden sich meist die Stammverzeichnisse der Benutzer. So könnten einige Verzeichnisse der nächsten Ebene sein (Verzeichnisnamen werden durch Schrägstriche `"/"` getrennt)

`/usr/lib`, `/usr/bin`, ..., `/home/beethoven`

Hier wäre z.B. `/home/beethoven` das Stamm- (Home-)Verzeichnis des Benutzers `beethoven`. In `/usr/lib` befinden sich in der Regel die Systembibliotheken und in `/usr/bin` Systemkommandos.

Sowohl Verzeichnis- als auch Filenamen können aus alphanumerischen (Groß- und Kleinbuchstaben) und Sonderzeichen bestehen und mehr oder weniger beliebig lang sein. Für die Filenamen verwendet man gern die Form

name.*extension*

wobei die Extension Aufschluß über den Inhalt des Files gibt, z.B.

```
eroica.mp3
```

Das hat den Vorteil, daß man bei vielen Systemprogrammen, die eine Standard-Extension erwarten, diese oft gar nicht angeben muß bzw. sich die Programme je nach bearbeitetem Filetyp verschieden verhalten können. So erwarten etwa der F- und C-Compiler, daß Programme die Extension `.F` oder `.c` haben, und ein guter Editor wird verschieden Formatierungshilfen zur Verfügung stellen, je nachdem, ob man ein Computerprogramm oder ein TEX-Manuskript (Extension `.tex`) einliest.

Die vollständige Angabe eines Filenamens besteht aus dem Pfad (dem Weg im Verzeichnisbaum, auf dem man von der Wurzel aus das File erreicht) und dem eigentlichen Filenamen

```
/home/beethoven/mp3/symphonies/eroica.mp3
```

Für einige spezielle Pfade bzw. Verzeichnisse sind Abkürzungen definiert, nämlich

```
.
```

für das gerade eingestellte Verzeichnis

```
..
```

für das Oberverzeichnis eines Verzeichnisses und

```
~
```

für das Home-Verzeichnis eines Benutzers. So könnte der Benutzer `beethoven` das File im letzten Beispiel auch mit

```
~/mp3/symphonies/eroica.mp3
```

ansprechen.

Die Zugriffsrechte auf ein File oder Verzeichnis werden in einer File-Liste durch eine zehnstellige Buchstabenkombination der Form

```
-rwxrwxrwx
```

dargestellt. Dabei symbolisieren die Stellen 2–4 die Rechte des Benutzers `u`, die Stellen 5–7 die Rechte der Gruppe `g`, der der Benutzer angehört, und die letzten drei Stellen die Rechte derer, die nicht der Gruppe angehören ("others" `o`). `r` und `w` stehen für Lese- und Schreibzugriff; `x` (execute) beim Filenamen eines Programms oder Scripts bedeutet, daß es ausgeführt werden darf. Besteht eines der Rechte nicht, steht statt des entsprechenden Buchstaben ein `"-"`. Ein `d` an der ersten Stelle heißt, daß es sich um den Namen eines Verzeichnisses handelt.

```
-rw-r-----
```

ist also der Code eines Files, das nur der Benutzer (Eigentümer) lesen und schreiben und die Gruppe lesen darf.

Shell

Von `sh` bis `bash`

Als "Shell" bezeichnet man jenes Systemprogramm, mit dem der Benutzer kommuniziert. Die Shell nimmt die Befehle des Benutzers entgegen und führt sie, wenn es sich um eingebaute Befehle handelt, selbst aus oder gibt sie an andere Programme des Betriebssytems weiter. Wenn nicht gerade ein anderes Programm die Kontrolle übernommen hat, befindet man sich also während jeder Arbeitssitzung, vom "Login" bis zum "Logout", ständig in Kontakt mit der Shell. Die Shell ist, neben anderen Aufgaben, auch für die Interpretation der Wildcards (`"*"` und `"?"`) zuständig sowie für die Übergabe von Umgebungsvariablen an System- und Benutzerprogramme.

Auf jedem Unix-Rechner gibt es die Bourne Shell `sh`, die aber eher für die Interpretation von Shell-Scripts als für das interaktive Arbeiten geeignet ist. Für letzteres stehen, je nachdem, um welche Variante von Unix es sich handelt, eine Reihe von etwas komfortableren Shells wie `csh`, `ksh`, `bash`, `tcsh`, usw. zur Auswahl. (Die Shell kann auch tatsächlich individuell gewählt werden.) Auf den derzeit populären Linux-Systemen ist `bash` die Standard-Shell. Alle Shells sind zwar in den Grundzügen ähnlich, aber in den Details doch so verschieden, daß es schwierig ist, sich in einer ungewohnten Shell zurecht zu finden.

Die `tcsh`

Wenn in diesem Skriptum Shell-spezifische Kommandos angegeben werden, beziehen sie sich in der Regel auf die `tcsh`. Die `tcsh` bietet u.a. folgende Annehmlichkeiten:

- Editieren der Kommandozeile: Der Text der Kommandozeile kann vor dem Absetzen des Kommandos mit Hilfe der Pfeil- und Löschtasten korrigiert und modifiziert werden.
- Rückholen früherer Kommandos: Je nach Einstellung befinden sich die letzten 20 oder mehr Kommandos in einem Buffer, aus dem sie mit Hilfe der ↑-Taste zurückgeholt werden können. Das Modifizieren "alter" Kommandos ist meist viel effizienter als das Eintippen neuer.

- Ergänzung von Kommandos: Um die Eingabe langer Namen zu vermeiden, genügt es bei Kommandos und Programmen im Suchpfad oder File- und Verzeichnisnamen, die ersten paar Zeichen zu tippen und dann die <Tab>-Taste zu drücken. Die `tcsh` versucht dann, das korrekte Kommando oder den Filenamen zu ergänzen bzw. bietet, wenn die Ergaänzung nicht eindeutig ist, eine Auswahl von Alternativen an.

Umgebungsvariable und Startup-Files

Auf einem Unix-System sind typischerweise einige Dutzend Umgebungsvariable gesetzt. Manche sind sytemweit definiert, andere wieder benutzerspezifisch. Zu den wichtigsten Umgebungsvariablen gehören `HOME`, das Stammverzeichnis des Benutzers, `PATH`, der Suchpfad, auf dem nach ausführbaren Programmen gesucht wird, und `DISPLAY`, das Gerät, auf das graphische Ausgaben gesendet werden.

Umgebungsvariable setzt man in der `tcsh` mit dem Befehl

`setenv` *env_variable value*

(set environment variable), den Wert einer Variable stellt man mit

`printenv` *[env_variable]*

(print environment variable) fest. Ein

`printenv`

ohne Argument zeigt eine Liste aller definierten Umgebungsvariablen.

Umgebungsvariable und sonstige Vereinbarungen kann man zwar jederzeit interaktiv setzen, Definitionen, die man bei jeder Arbeitssitzung braucht, schreibt man aber besser in eines der Startup-Files, die bei jedem Login bzw. Starten einer Shell ausgeführt werden. In der `tcsh` sind das die Files im `HOME`-Verzeichnis des Benutzers

`.cshrc` oder, wenn vorhanden, `.tcshrc`
`.login`

(in dieser Reihenfolge).

Die allerwichtigsten Befehle

Erste Hilfe

Bei jedem Programm, das man benützt oder aufruft, ist es am wichtigsten zu wissen, wie man es wieder verläßt bzw. abbricht, wenn es außer Kontrolle geraten ist. Dies gilt natürlich auch für das Programm “Shell” selbst, also für die Arbeitssitzung überhaupt. Während das Einloggen meist darin besteht, daß man Accountnamen und Passwort in eine Eingabemaske oder auf eine Eingabeaufforderung eingibt, beendet man die Sitzung in der `tcsh` mit dem Befehl

`logout`

oder

```
exit
```

(In anderen Shells kann auch `^D` die gewünschte Wirkung haben.)

Innerhalb der Shell bricht man ein Kommando oder Programm, das man im Vordergrund gestartet hat, mit

```
^C
```

ab. Prozesse, die im Hintergrund laufen, kann man mit

`kill -9` *process_id*

beenden, wobei man die Prozeß-Nummer *process_id* z.B. mit dem `ps`-Kommando ermitteln kann.

Die Dokumentation der Befehle und Prozeduren des Betriebssystems sowie über zusätzlich installierte Software befindet sich (in etwas archaischer Form) auf den sogenannten `man`-Pages. Informationen über das Thema *topic* kann man mit

`man` *topic*

erhalten. `man`-Pages werden häufig seitenweise über einen Pager (s. unten) dargestellt, sodaß man u.U. vor- und rückwärts blättern oder nach Zeichenketten suchen kann.

Speziell auf Linux-System findet man Dokumentation auch im hierarchisch organisierten `info`-Informationssystem. Der Einstieg erfolgt mit

`info` *topic*

Kurzinformationen oder die Syntax eines Kommandos kann man oft auch mit

command `-h`

oder

command `--help`

ermitteln.

Die allgemeine Struktur der Befehle ist

command [options] [arguments]

wobei *arguments* die etwaigen Argumente sind und *options* zusätzlich Optionen, die das Verhalten des Befehls modifizieren oder genauer definieren. Diese Optionen bestehen meist aus einem Minuszeichen und einer einbuchstabigen Abkürzung (wie `-h` für "Help"), können aber auch von der Form *option parameter* sein.

Manipulation von Files und Verzeichnissen

Aktuelles Verzeichnis anzeigen
Um nicht immer vollständige Pfadnamen angeben zu müssen, kann man ein Verzeichnis zum aktuellen Arbeitsverzeichnis machen. Welches das aktuelle Verzeichnis ist, kann man mit dem Befehl

```
pwd
```

(print working directory) feststellen.

Verzeichnis wechseln
Das Arbeitsverzeichnis wechselt man mit

`cd` *[directory]*

(change directory). *directory* kann ein absoluter oder relativer Pfad (d.h. relativ zum derzeitigen Arbeitsverzeichnis) sein. Spezialfälle sind

```
cd ..
```

(wechselt in das Verzeichnis über dem derzeit aktuellen) und

```
cd ~
```

(wechselt in das Home-Verzeichnis). Ein

```
cd
```

ohne Argument hat dieselbe Wirkung wie `cd ~`.

Neues Verzeichnis anlegen
Ein neues Verzeichnis legt man mit

`mkdir` *directory*

(make directory) an. *directory* kann wieder eine absolute oder relative Pfadangabe sein. Der häufigste Fall ist wohl der, daß man ein Unterverzeichnis unter dem aktuellen Arbeitsverzeichnis anlegen will.

Verzeichnis löschen
Ein (leeres) Verzeichnis löscht man mit Hilfe von

`rmdir` *directory*

(remove directory). Um mit diesem Befehl ein Verzeichnis löschen zu können, darf es keine Files oder Unterverzeichnisse mehr enhalten. Verzeichnisse samt Inhalt löscht man mit Hilfe des Befehls `rm` (s. dort).

Inhalt eines Verzeichnisses anzeigen
Die in einem Verzeichnis enthaltenen Files zeigt der Befehl

`ls` *[file]*

(list) an. *file* ist ein File- oder Verzeichnisname und kann Wildcards enthalten. Ist es ein Verzeichnisname, so werden alle Files im Verzeichnis angezeigt. Ist *file* ein Filename, so wird das File, bzw. im Fall von Wildcards die Files angezeigt, auf die das Muster paßt. Ein

```
ls
```

ohne Argument zeigt den Inhalt des aktuellen Arbeitsverzeichnisses. Die wichtigesten Optionen sind `-l` ("lange" Liste, ein Eintrag pro Zeile) und `-a` (zeigt auch die "Hidden Files", deren Namen mit einem `.` beginnen und die man normalerweise nicht sieht). Die Optionen können auch kombiniert werden, z.B. produziert

```
ls -al ..
```

eine lange Liste aller (auch der versteckten) Files im Verzeichnis über dem Arbeitsverzeichnis.

Files kopieren
Der Befehl

`cp` *source destination*

(copy) erstellt eine Kopie von *source* unter dem Namen *destination*. Ist *destination* ein Verzeichnis, so erhält die Kopie den Namen des Originals. In diesem Fall können auch mehrere *sources* angegeben und Wildcards verwendet werden

`cp` *source_1 source_2 ...* *directory*

Umbenennen von Verzeichnissen und Files
Files und Verzeichnisse können mit dem Befehl

`mv` *source destination*

(move) umbenannt bzw. im Verzeichnisbaum "bewegt" werden. Sind *source* und *destination* beides Files oder beides Verzeichnisse, dann erhält *source* den neuen Namen *destination*. Da sowohl alter als auch neuer Name Pfadangaben enthalten dürfen, können damit Objekte gleichzeitig von einer Stelle im Verzeichnisbaum an eine andere verlagert werden. Insbesondere werden beim `move` eines Verzeichnisses dessen Inhalt und eventuelle Unterverzeichnisse mitverlagert. In der Form

`mv` *source_1 source_2 ...* *directory*

(wobei *source_1*, *source_2*, usw. Wildcards enthalten können) ist es auch möglich, ganze Gruppen von Files auf einmal in das Verzeichnis *directory* zu verschieben.

Löschen von Files
Ein oder mehrere Files löscht man mit dem Befehl

`rm` *file_1 file_2 ...*

(remove), wobei die Filenamen wieder Wildcards enthalten dürfen. Insbesondere beim Löschen von Gruppen von Files empfiehlt sich die Option `-i` (inquire)

`rm -i` *file_1 file_2 ...*

da dann für jedes zu löschende Objekt eine Bestätigung gegeben werden muß. Den gegenteiligen Effekt hat die Option `-f` (force). Ganze Verzeichnisse samt Inhalt kann man schließlich mit der Option `-r` (recursive) löschen, z.B. in der Form

`rm -fr` *directory_1 directory_2 ...*

Inhalt eines Files ansehen
Will man den Inhalt eines Files nur ansehen, ohne es zu verändern, so kann man dazu das Kommando

`cat` *file_1 file_2 ...*

(concatenate) verwenden. Dieser Befehl reiht die Files *file_1*, *file_2*, usw. aneinander und gibt das Resultat am Bildschirm aus. Das hat allerdings bei längeren Files den Nachteil, daß der Inhalt sehr schnell über den Bildschirm läuft und man nur die letzten paar Zeilen sieht.

Daher gibt es unter jedem Betriebsystem sogenannte Pager. Das sind Programme, die es erlauben, Files seitenweise anzusehen. Der Standard-Pager unter Unix ist

`more` *file*

Daneben gibt es aber meist auch noch etwas komfortablere Programme wie

`less` *file*

oder

`most` *file*

mit denen man sowohl vorwärts als auch rückwärts blättern (mit den Tasten `u`, `d`, `^u`, `^d` oder <Page-Up>, <Page-Down>) oder nach Zeichenketten suchen kann (nach Eingabe von `"/"`). Der Pager wird mit `q` (quit) wieder verlassen.

Ein- und Ausgabeumlenkung, Pipes

Die meisten Systemprogramme sowie viele Anwendungen kommunizieren über die Standard-Ein/Ausgabekanäle von Unix

```
stdin
stdout
stderr
```

Der Standard-Eingabekanal `stdin` ist normalerweise mit der Tastatur verknüpft, der Standard-Ausgabekanal `stdout` und der Ausgabekanal für Fehlermeldungen `stderr` mit dem Bildschirm. Viele Programme verhalten sich wie Filter, d.h. sie lesen ihren Input von `stdin`, verarbeiten ihn und geben den Output auf `stdout` aus.

Die Standard-Ein/Ausgabekanäle können auf einfache Weise mit Hilfe der Klammersymbole `"<"`, `">"` und `">>"` umgelenkt werden. So bedeutet

command `<` *file*

daß der Input für den Befehl oder das Programm *command* nicht von der Tastatur sondern vom File *file* kommen soll. Umgekehrt wird mit

command `>` *file*

der Output statt auf den Schirm auf ein File ausgegeben. Bei der Variante

command `>>` *file*

wird ein eventuell existierendes File nicht überschrieben, sondern der Output von *command* wird an das Ende von *file* angefügt. Ein- und Ausgabe können auch gleichzeitig umgelenkt werden

command `<` *input_file* `>` *output_file*

wobei die Reihenfolge von Input/Ouput auch umgekehrt werden kann.

Manchmal möchte man auch erreichen, daß ein Befehl, der als Argument einen Filenamen verlangt, statt von einem File von der Tastatur liest. Dafür gibt es das Symbol `"-"`

command `-`

Um beispielsweise Text in das File *file* zu schreiben, könnte man das Kommando `cat` aufrufen

`cat >` *file* `-`

den gewünschten Inhalt von *file* auf der Tastatur eintippen und die Eingabe mit `^d` (der End-of-File Marke) abschließen.

Pipes sind ein Mechanismus, mit dessen Hilfe zwei oder mehrere "Filterprogramme" gleichzeitig gestartet werden können, wobei jeweils der Standard-Ausgabekanal eines Programms in den Standard-Eingabekanal des nächsten geleitet wird. Pipes werden durch den senkrechten Strich `"|"` symbolisiert

command_1 `|` *command_2* `|` ...

Beispiel: Die Kombination

```
ls -l | grep "Mar 30"
```

erzeugt eine Liste aller Files im aktuellen Verzeichnis, die am 30. März zuletzt modifiziert wurden. (`grep` ist ein Befehl, der in einer Liste von Files oder der Standard-Eingabe—die in diesem Fall mit der Standard-Ausgabe des `ls`-Kommandos identifiziert wird—nach einer Zeichenkette sucht und alle Zeilen ausgibt, in der sie vorkommt).

Editieren und Compilieren von einfachen Programmen

Emacs

Um ein Computerprogramm bzw. allgemein ein Text-File neu zu schreiben oder ein bestehendes zu modifizieren, braucht man ein spezielles Textverarbeitungsprogramm, den sogenannten "Editor". Wie bei anderer Unix-Software auch, gibt es eine ganze Reihe von Alternativen, und welchen Editor man verwendet, hängt einerseits von der Anwendung ab, ist aber vielfach persönliche Geschmackssache.

Der Standard-Editor von Unix, `vi`, ist—wenigstens in der Grundversion—nicht übermäßig komfortabel. Ein alternativer Standard-Editor, der ebenfalls auf jeder Unix-Installation vorhanden sein sollte, ist `emacs`. Beim `emacs` handelt es sich eigentlich nicht um einen Editor im herkömmlichen Sinn, sondern um einen Lisp-Interpreter, also ein recht umfangreiches Softwarepaket, aus dem heraus man nicht nur Programme schreiben, compilieren und ausführen, sondern auch Mail lesen, surfen, auf Files anderer Maschinen im Netz zugreifen kann usw. Zusätzlich ist das Verhalten des `emacs` weitgehend anpaß- und programmierbar.

Selbst wenn man nicht die Absicht hat, diese Möglichkeiten auch nur ansatzweise auszuschöpfen, ist der `emacs` aber schon für das bloße Schreiben von Programmen sehr hilfreich. Abhängig von der Extension des bearbeiteten Files (also etwa `.c` für ein C-Programm) läuft er nämlich in einem speziellen Modus, in dem er den Benutzer bei der Formatierung des Programms und der Vermeidung von Fehlern unterstützt. So wird z.B. automatisch eingerückt, beim Eingeben einer schließenden Klammer die zugehörige öffnende angezeigt, Schlüsselwörter werden hervorgehoben usw.

Der Editor wird aufgerufen mit

`emacs` *[file]*

Falls man unter einer graphischen Umgebung (X-Windows) arbeitet, wird ein eigenes Fenster geöffnet. Von der ursprünglichen Version von *file* wird eine Kopie unter dem Namen *file~* angelegt, auf die man zurückgereifen kann, wenn beim Editieren etwas schiefgegangen ist.

Beim Eingeben oder Modifizieren des Programms kann man die üblichen Lösch-, Pfeil- und Navigationstasten benützen. Das Suchen und Ersetzen von Zeichenketten, Ausschneiden und Einsetzen von Blöcken, Abspeichern des Buffers usw. erfolgt mit verschiedenen Tastenkombinationen, deren wichtigste in der "Emacs/Jed-Kurzanleitung" zusammengefaßt sind. Eine

Besonderheit ist die Wirkung der <Tab>-Taste: Es wird bei Drücken von <Tab> nicht nur die Zeile, in der der Cursor steht, relativ zur vorhergehenden ausgerichtet (also z.B. ein- oder ausgerückt), sondern u.U. auch eine Sprachkonstruktion ergänzt. Drückt man etwa in dem folgenden Fragment einer `do`-Schleife in F

```
loop: do i=1,imax
  ...
  end
```

nach der Eingabe von `end` auf <Tab>, so wird nicht nur das `end` nach links ausgerückt, sondern zu

```
end do loop
```

ergänzt.

Vom `emacs` gibt es zahlreiche Clones, u.a. den Editor `jed` mit der X-Version `xjed`. `jed/xjed` (s. `http://space.mit.edu/~davis/jed/`) sind wesentliche weniger umfangreich und haben auch nicht die volle Funktionalität des `emacs`, können aber eine Reihe weiterer Editoren emulieren und sind auf praktisch alle Betriebssysteme portiert worden.

F- und C-Compiler

Hier wird zunächst nur der Fall besprochen, daß sich Haupt- und Unterprogramme (Prozeduren) in ein und demselben File befinden. Dann ist der Befehl zum Übersetzen eines F-Programms

`F` *[options] source_file*

Einschließlich der wichtigsten Optionen lautet der Compiler-Aufruf meist

`F -O -o` *executable source_file*

Dabei steht die Option `-O` für "optimize", während `-o` *executable* angibt, wie der Filename des ausführbaren (compilierten und gelinkten) Programms lauten soll. Gibt man die letztere Option nicht an, heißt des ausführbare Programm standardmäßig `a.out`. Soll also ein Programm mit dem Namen `prog1.F` übersetzt werden, kann man z.B. an

```
F -O -o prog1 prog1.F
```

und das Programm mit

```
prog1
```

starten. Treten während der Compilation Fehler auf, so wird der Programmtext samt den Fehlermeldungen in ein File mit dem Namen `F.err` geschrieben. Der Compiler erzeugt als Nebenprodukte ein Unterverzeichnis `f_files`, ein File `f_moddir.tmp` sowie Modul-Files mit der Extension `.mod`, die nach erfolgreicher Compilation alle gelöscht werden können.

Ein C-Programm compiliert man analog mit

`cc` *[options] source_file*

bzw. wenn es sich um den GNU C-Compiler handelt (der aber meist auch durch `cc` aufgerufen werden kann),

`gcc` *[options] source_file*

Mit den wichtigsten Optionen kann das lauten

`gcc -O`*n* `-o` *executable* `-l`*library_1* `-l`*library_2* ... *source_file*

Für die Optimierung können hier verschiedene Stufen n=0, 1, ... angegeben werden.

Die Schwierigkeit gegenüber F besteht darin, daß bei C häufig auch jene Bibliotheken, in denen sich Standardprozeduren befinden, beim Linken explizit angegeben werden müssen. Das erfolgt mit der Option `-l` (library). Diese Bibliotheken stehen in der Regel in

`/usr/lib`

und haben Namen der Form

`lib`*name*`.a`

Da die Namen aller Bibliotheken mit `"lib"` beginnen und mit `".a"` oder `".so"` enden, sind diese Teile redundant, und bei der Option `-l` muß nur *name* angegeben werden.

Benützt z.B. das Programm `prog2.c` trigonometrische Funktionen, die in der Mathematik-Bibliothek

`/usr/lib/libm.a`

stehen, so könnte es mit

```
gcc -O2 -o prog2 -lm prog2.c
```

übersetzt und gelinkt werden.