# Betriebssystem Linux und Programmiersprachen: Eine Einführung

Michael Wegner (Linux)
Joachim Puls (Fortran 90)

Inhalt:



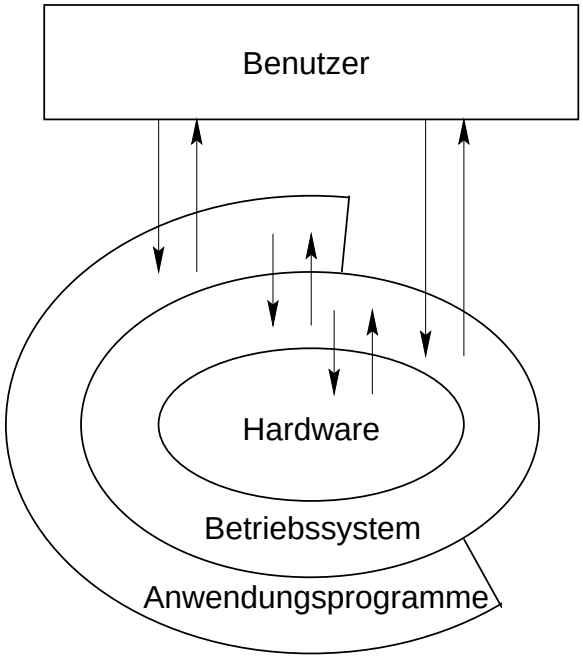
Zusatzmaterial

- Prinzip `vi`: `vi_brief.pdf`
- Referenz für `vi`: `vi_reference.pdf`
- Referenz für `emacs`: `emacs_reference.pdf`

# 1 Allgemeines zum Betriebssystem UNIX/Linux

## Einordnung von UNIX/Linux

UNIX ist ein *Multi-User/Multi-Tasking-Betriebssystem* und existiert in vielen verschiedenen Versionen ("Derivaten"): Solaris, AIX, XENIX, HP-UX, SINIX, **Linux**.




**Betriebssystem (OS)**: Summe der Programme, die zum *Betrieb* eines Rechners *notwendig* sind und die Abwicklung von Anwendungsprogrammen steuern und überwachen.



## Wesentliche Merkmale

### UNIX

- ist ursprünglich in der Programmiersprache C geschrieben und deshalb eine klassische Plattform für C-Programme. UNIX besitzt ausgereifte Umgebungen zur Programmentwicklung (C, C++, Java, Fortran, ...).

- wird vor allem in wissenschaftlich-technischen Anwendungsbereichen sowie auf Großrechnern und Workstations genutzt, ist durch **Linux** in den letzten Jahren aber auch für klassische PC-Anwendungsgebiete interessant geworden.

- eignet sich hervorragend zum Einsatz in Netzwerken. Größere Systeme bzw. Netzwerke erfordern einen Administrator.

- bietet zur Lösung der meisten Aufgaben jeweils mehrere Alternativen. Die zahlreichen (mehr als bei jedem anderen OS) Kommandos sind kurz und flexibel.

- kann über eine graphische Benutzeroberfläche (*X-Window-System*) bedient werden, ist aber ursprünglich kommandozeilenorientiert.

- **Linux** wird über das Internet oder in verschiedenen *Distributionen* (S.u.S.E., Fedora, Debian etc.) vertrieben. Mittlerweile existiert eine Vielzahl von *Direktstart-(Live-)Systemen*, die ohne

Installation direkt von CD (oder anderen bootfähigen Speichermedien) gestartet werden (Knoppix, Ubuntu, ...), u.a. auch ansprechende "Miniversionen" (<100 MB), die für den Start von USB-Sticks vorgesehen sind (z.B. Puppy). Der Quellcode von Linux ist frei.

## Literatur

- **Gilly,D. u.a.**: *UNIX in a Nutshell.* O'Reilly, Köln. 1998 (1.Auflage). DM 49,-

- **Wielsch,M.**: *Das große Buch zu UNIX.* Data Becker, Düsseldorf. 1994 (1.Auflage). DM 79,-

- unzählige weitere Bücher

- **online-tutorial (sehr empfehlenswert)**
  `http://www.stickybit.de/wissen/computer/anleitungen/unix/index.htm`

# 2 Erste Schritte am Rechner

## Benutzer, Anmelden und Abmelden

Als *Multiuser*-Betriebssystem kann UNIX mehrere Benutzer gleichzeitig bedienen, die allerdings in jedem Fall eine *Benutzerkennung (account)* benötigen.

Jeder Benutzer hat eine persönliche Umgebung (*Home-Directory, Shell*), auf die nur er selbst Zugriff hat.

Systemintern wird ein Benutzer identifiziert durch eine eine Benutzernummer (*user ID, UID*) sowie eine Gruppenzugehörigkeit mit Gruppennnummer (*group ID, GID*).

Es gibt zwei Typen von Benutzern:

- die 'normalen' Benutzer mit *eingeschränkten* Rechten und den

- Systemadministrator (*root*) mit allen Privilegien. Dieser ist verantwortlich für Installation und Konfiguration des Systems, Wartung, Benutzerverwaltung usw.

Jeder Benutzer muß sich an- und wieder abmelden (*login/logout*). Jede Benutzerkennung ist dabei durch ein *Paßwort* geschützt.

**Übung:**

*Melden Sie sich mit Ihrer Benutzerkennung am System an!*

## Graphische Benutzeroberfläche

UNIX ist ursprünglich kommandozeilenorientiert. Das *X-Window-System* ermöglicht allerdings eine komfortable Bedienung über eine fensterorientierte graphische Oberfläche – ähnlich wie in anderen Betriebssystemen.

Verantwortlich für die Verwaltung und Darstellung der einzelnen Fenster ist der *Windowmanager*. Jeder Windowmanager – eine ganze Reihe unterschiedlicher Versionen sind gebräuchlich – zeichnet sich durch ein eigenes *Look and Feel* (Aussehen der Fenster, Steuerelemente usw.) aus.

Einfache Windowmanager sind z.B.:

- `twm`: sehr einfach und ressourcensparend
- `fvwm, fvwm2`: komfortabler und trotzdem einfach
- `mwm`: Motif Windowmanager, weit verbreitet

Fast alle Linux-Distributionen bieten daneben graphische Oberflächensysteme (*Desktop Environments*) wie **KDE** oder GNOME an, deren Funktionsumfang weit über die Eigenschaften eines einfachen Windowmanagers hinausgeht.



Kommando `xterm`

---

**Syntax:**

```
xterm [Optionen]
```

Trotz graphischer Oberflächen ist UNIX ohne Möglichkeit zur direkten Eingabe von Kommandos praktisch nicht bedienbar. Mindestens ein Terminal-Fenster muß deshalb immer offen sein. Meist läßt sich das mit Hilfe des Windowmanagers erreichen.

Mit dem Befehl `xterm` können dann beliebig viele weitere Kommandofenster geöffnet werden.

Im allgemeinen haben alle UNIX-Kommandos eine Vielzahl von *Optionen*, die meist mit - beginnen und von denen bei Vorstellung der folgenden Kommandos jeweils nur die wichtigsten aufgeführt sind.

**Beispiel:**

```
wegner@arber:~ > xterm -geo 80x40 -fn 10x20
```

Das Kommando `xterm` wird mit zwei Optionen `-geo, -fn` aufgerufen, die in diesem Fall zusätzliche Argumente (Fensterbreite, -höhe bzw. Fontgröße) benötigen.

**Übung:**

1. *Öffnen Sie über KDE ein Terminal-Fenster!*
2. *Starten Sie daraus ein* `xterm`*!*

## Kommando `man`

### Syntax:

```
man [Sektion] Kommando
man -k Ausdruck
```

Aufruf der zum angegebenen Kommando gehörigen Seite des Online-Handbuches. `man -k` sucht nach Manual-Seiten für Kommandos in denen `Ausdruck` vorkommt. Ein Manual-Text besteht üblicherweise aus folgenden Abschnitten:

- `SYNOPSIS` Kommandosyntax
- `DESCRIPTION` Beschreibung der Wirkung des Kommandos
- `FILES` Dateien, die verändert bzw. vorausgesetzt werden
- `OPTIONS` Kommandooptionen, falls vorhanden
- `EXAMPLE` Beispiel(e) zur Anwendung, meist sehr spärlich
- `BUGS` Fehler, falls bekannt
- `SEE ALSO` Querverweise auf andere Kommandos im gleichen Zusammenhang

### Übung:

*Lassen Sie sich Informationen zum Kommando* `xterm` *anzeigen!*



## Kommando `passwd`

### Syntax:

```
passwd
```

Setzt ein neues Paßwort.

Paßwörter sollten aus einer Kombination aus Buchstaben und Ziffern bestehen, die nicht anhand von Wörterbüchern o.ä. durch systematisches Suchen entschlüsselt werden kann.

Das Kommando zum Setzen des Paßwortes und die dabei zu beachtenden Konventionen (Länge, Anzahl Ziffern/Buchstaben) *können von System zu System variieren!!!* Folgendes Beispiel entspricht dem "normalen" Vorgehen, z.B. für die Rechner des CIP-Pools (nicht jedoch für die Rechner der USM).

### Beispiel:

```
wegner@arber:~ > passwd
Changing password for wegner
Old password:
Enter the new password
  (minimum of 5, maximum of 8 characters)
Please use a combination
  of upper and lower case letters and numbers.
New password:
Re-enter new password:
Password changed.
----> Ihr neues Passwort ist in 5 Minuten
      im gesamten Pool aktiv! <-----
Connection to 141.84.136.1 closed.
wegner@arber:~ >
```

Kommando `who, whoami`

---

**Syntax:**

```
who
whoami
```

`who` zeigt für alle Benutzer, die am System angemeldet sind:

- Benutzername
- Terminal, an dem der jeweilige Benutzer arbeitet
- Zeitpunkt der Anmeldung

`whoami` ist selbsterklärend.

**Beispiel:**

```
wegner@arber:~ > whoami
arber!wegner   pts/5    Oct 20 12:45
```



## Arbeiten an externen Terminals

Zur Anmeldung an einem entfernten Rechner ist jeweils die IP-Adresse entweder numerisch oder als Rechnername anzugeben. In lokalen Netzwerken (CIP-Pool) genügt der Rechnername. Die alten Befehle ( "r-Befehle, z.B. `rlogin`) gelten dabei als unsicher, und man benützt in den letzten Jahren fast ausschließlich die "secure" Befehle, die die übertragenen Daten verschlüsseln.

Kommando `ssh`

---

**Syntax:**

```
ssh -X -l username hostname
ssh -X username@hostname
```

Ermöglicht die Anmeldung an einem beliebigen über eine IP-Adresse lokalisierbaren Rechner, für den man eine Benutzerkennung besitzt. Das Abmelden erfolgt mit `exit`, `Ctrl D` oder `logout`.

**Beispiel:**

```
wegner@arber:~ > ssh -l -X wegner lxsrv1.lrz-muenchen.de
wegner's password:
Last login: Sun Oct 22 ...
**********************
Mitteilungen
**********************
wegner@lxsrv1:~ > logout
Connection to lxsrv1.lrz-muenchen.de closed.
```

ODER (falls Verbindung mit "eigenem" Cluster)

**Beispiel:**

```
wegner@arber:~ > ssh -X wegner@arber
Last login: Sun Oct 22 ...
etc. (keine Passwort-Abfrage)
```

Die `secure shell` hat außerdem den Vorteil, dass der Rechner `hostname` X-Anwendungen auf dem aktuellen Terminal auszuführen kann, ohne dass das Kommando `xhost` benötigt wird. Ggf. ist die Option `-X` anzugeben.

Kommando `scp`

---

Um Dateien von einem Rechner auf einen anderen zu kopieren, verwendet man den Befehl `scp` ("secure copy", siehe auch `cp`).

**Syntax:**

```
scp datei1 username@hostname:datei2
scp username@hostname:datei1 datei2
```

Der erste Befehl kopiert die lokale Datei `datei1` auf den Fremdrechner unter dem Namen `datei2`, der zweite Befehl zeigt den umgekehrten Fall. Man beachte den Doppelpunkt. `scp -r` erlaubt das *rekursive* Kopieren ganzer Verzeichnisse, vgl. `cp -r`.

# 3 Dateisystem

## Logischer Aufbau, Dateitypen

*"In UNIX ist alles eine Datei."*

Als *Dateisystemobjekte* können auftreten

- 'normale' (Text-)Dateien
- Verzeichnisse
- ausführbare Dateien (Binärdateien oder *Shell-Skripte*)
- Gerätedateien
- *Pipes*
- symbolische *Links* (Verweise auf Dateien)

Alle Dateien bzw. Dateisystemobjekte ordnen sich ein in einen hierarchischen *Verzeichnisbaum* mit genau einem *Wurzelverzeichnis '/'*.

Im Gegensatz zum Windows-Dateisystem kennt das UNIX-Dateisystem keine unterschiedlichen Laufwerke. Alle physikalisch vorhandenen Speichermedien (Platten, CDROM, Floppy) verbergen sich hinter bestimmten Verzeichnissen innerhalb des Verzeichnisbaums (normalerweise unter `/dev`).

Dateinamen bestehen aus einer Zeichenfolge aus Buchstaben, Ziffern und bestimmten Sonderzeichen und können weder Leerzeichen noch *slash* enthalten.

Zeichen, die von der *Shell* gesondert interpretiert werden, sollten vermieden werden.

Eine Datei wird innerhalb des Verzeichnisbaums durch einen *absoluten* oder *relativen Pfadnamen* lokalisiert. Ein absoluter Pfadname besteht aus allen Verzeichnissen und dem Dateinamen und beginnt immer mit /.

Die Tilde steht für das Home-Verzeichnis.

## Kommando `pwd`

**Syntax:**

```
pwd
```

Zeigt das aktuelle Verzeichnis an.

**Beispiel:**

```
wegner@arber:~ > pwd
/home/wegner
wegner@arber:~ >
```

**Übung:**

*Lassen Sie sich das aktuelle Verzeichnis anzeigen!*



## Kommando `cd`

**Syntax:**

cd [Verzeichnis]

Wechselt in das angegebene Verzeichnis bzw. ins eigene Home-Verzeichnis, wenn kein Parameter angegeben wird.

Wie in DOS/Windows bezeichnet ".." das übergeordnete und "." das aktuelle Verzeichnis.

**Beispiel:**

```
wegner@arber:~ > cd /home/puls
wegner@arber:/home/puls > pwd
/home/puls
wegner@arber:/home/puls > cd ..
wegner@arber:/home > pwd
/home
wegner@arber:/home > cd
wegner@arber:~ > pwd
/home/wegner
wegner@arber:~ >
```

**Übung:**

*Wechseln Sie ins Verzeichnis (*`/opt/intel_fc_81`*) und wieder zurück in Ihr Home-Verzeichnis! (→ Dateinamenvervollständigung mit* `TAB`*)*

*Überprüfen Sie den Erfolg jeweils mit* `pwd`*!*

## Suchmuster für Dateinamen

Die *Shell* ist im Prinzip ein eigenes Programm, das die Interpretation eingegebener Kommandos übernimmt. Haben diese Kommandos Parameter, die Dateinamen sind, lassen sich durch *Suchmuster*, die durch die Shell *expandiert* werden, mehrere Dateien gleichzeitig ansprechen. In jedem Fall wird die Dateinamensexpansion *vor* der Ausführung des Kommandos vorgenommen.

| Sonderzeichen | Bedeutung |
|---|---|
| `*` | beliebige, auch leere Zeichenkette |
| `?` | ein beliebiges einzelnes Zeichen |
| `[...]` | ein Bereich von Zeichen |
| `[!...]` | negierter Bereich von Zeichen |

Kommando `ls`

---

**Syntax:**

```
ls [-alR] [Datei/Verzeichnis]
```

Zeigt Namen und evtl. Kenndaten von Dateien an bzw. listet den Inhalt des Verzeichnisses auf. Als Datei- und Verzeichnisnamen sind sowohl absolute als auch relative Pfadangaben erlaubt.



Wichtige Optionen:

`-a` Auch Dateien, die mit einem Punkt beginnen (versteckte Dateien), werden gelistet.

`-l` Anzeige im Langformat. U.a. Zugriffsrechte, Benutzer- und Gruppennummer, Zeitstempel, Größe usw.

`-R` Zu Verzeichnissen werden auch alle Unterverzeichnisse rekursiv angezeigt.

**Beispiel:**

```
wegner@arber:~ > ls
hello*  hello.cpp  hello.f90  nsmail/
wegner@arber:~ > ls -a
./           .bash_history  .netscape/  hello.cpp
../          .bashrc*       .ssh/       hello.f90
.Xauthority  .history       hello*      nsmail/
wegner@arber:~ > ls /var/X11R6
app-defaults/  bin/    lib@  sax/
scores/        xfine/  xkb/
wegner@arber:~ > ls .b*
.bash_history  .bashrc*
wegner@arber:~ > ls [a-h]*
hello*  hello.cpp  hello.f90
wegner@arber:~ > ls *.?[9p]?
hello.cpp  hello.f90
wegner@arber:~ >
```

**Übung:**

*Lassen Sie sich den vollständigen Inhalt Ihres Home-Verzeichnisses anzeigen!*

## Dateien kopieren, verschieben und löschen

Neben `ls` stehen eine ganze Reihe weiterer Kommandos zur Arbeit mit Dateien zur Verfügung, bei denen jeweils wieder mit Suchmustern gearbeitet werden kann.

Kommando `mkdir, rmdir`

---

**Syntax:**

```
mkdir Verzeichnis
rmdir Verzeichnis
```

`mkdir` legt ein leeres Verzeichnis an, `rmdir` löscht ein Verzeichnis. Das Verzeichnis muß vor dem Löschen leer sein.

**Beispiel:**

```
wegner@arber:~ > ls
hello*  hello.cpp  hello.f90  nsmail/
wegner@arber:~ > mkdir numerik
wegner@arber:~ > ls
hello*  hello.cpp  hello.f90  nsmail/  numerik/
wegner@arber:~ > rmdir numerik
wegner@arber:~ > ls
hello*  hello.cpp  hello.f90  nsmail/
wegner@arber:~ >
```

**Übung:**

*Legen Sie in Ihrem Home-Verzeichnis ein Verzeichnis* `uebung` *an!*



Kommando `cp`

---

**Syntax:**

```
cp [-r] Datei1 Datei2
cp [-r] Datei1 [Datei2 ...] Verzeichnis
```

Kopiert Dateien oder Verzeichnisse. Die Originaldatei bzw. das Originalverzeichnis bleibt erhalten.

Option:

`-r` Verzeichnisse werden rekursiv mit allen Unterverzeichnissen kopiert.

**Beispiel:**

```
wegner@arber:~ > ls
hello*  hello.cpp  hello.f90  nsmail/  numerik/
wegner@arber:~ > cp hello.cpp hello2.cpp
wegner@arber:~ > ls
hello*     hello.f90   nsmail/
hello.cpp  hello2.cpp  numerik/
wegner@arber:~ > cp hello.cpp numerik
wegner@arber:~ > ls numerik
hello.cpp
wegner@arber:~ >
```

**Übung:**

*Kopieren Sie die Dateien aus* `ubung0` *in Ihr Verzeichnis* `uebung`*!*

Kommando `mv`

---

## Syntax:

```
mv Datei1 Datei2
mv Datei1 [Datei2 ...] Verzeichnis
```

Umbenennen oder Verschieben von Dateien oder Verzeichnissen. Wenn der letzte Parameter ein Verzeichnis ist und existiert, wird verschoben, sonst umbenannt.

## Beispiel:

```
wegner@arber:~ > ls
hello*     hello.f90   nsmail/
hello.cpp  hello2.cpp  numerik/
wegner@arber:~ > mv hello2.cpp hello3.cpp
wegner@arber:~ > ls
hello*     hello.f90   nsmail/
hello.cpp  hello3.cpp  numerik/
wegner@arber:~ > ls numerik
hello.cpp
wegner@arber:~ > mv hello3.cpp numerik
wegner@arber:~ > ls
hello*  hello.cpp  hello.f90  nsmail/  numerik/
wegner@arber:~ > ls numerik
hello.cpp  hello3.cpp
wegner@arber:~ >
```

## Übung:

1. *Benennen Sie Ihr Verzeichnis* `uebung` *in* `uebung0` *um!*

2. *Verschieben Sie die Datei* `.plan` *aus* `uebung0` *in Ihr Home-Verzeichnis! Versuchen Sie auch, eine Datei ins Wurzelverzeichnis zu verschieben!*

Kommando `rm`

---

**Syntax:**

```
rm [-irf] Datei(en)
```

Löschen von Dateien oder Verzeichnissen. Ein nachträgliches Wiederherstellen der gelöschten Daten ist *nicht* möglich! `rm` sollte daher nur mit äußerster Vorsicht angewendet werden, denn z.B. `rm -rf *` löscht ohne Rückfrage rekursiv den gesamten Dateibaum unterhalb des aktuellen Verzeichnisses.

Optionen:

- `-i` Es wird erst nach vorheriger Sicherheitsabfrage gelöscht.
- `-r` Verzeichnisse werden rekursiv mit allen Unterverzeichnissen gelöscht.
- `-f` Unterdrückung aller Sicherheitsabfragen.

**Beispiel:**

```
wegner@arber:~/numerik > ls
hello.cpp  hello3.cpp
wegner@arber:~/numerik > rm -i hello3.cpp
rm: remove ‘hello3.cpp’? y
wegner@arber:~/numerik > ls
hello.cpp
wegner@arber:~/numerik >
```



## Rechtevergabe

Das UNIX-Dateisystem kennt drei verschiedene Zugriffsrechte:

- `r` Lesen: Erlaubt das Auslesen des Inhaltes von Dateien bzw. bei Verzeichnissen das Anzeigen von Verzeichnisinhalts.
- `w` Schreiben: Erlaubt das Verändern des Dateiinhaltes oder das Löschen der Datei. Um Dateien Anlegen oder Löschen zu können, muss auch das übergeordnete Verzeichniss eine Schreibberechtigung haben!
- `x` Ausführen: Ermöglicht für ausführbare Binärdateien (Kommandos, Programme) und Shell-Skripte das Starten von der Kommandozeile. Verzeichnisse erlauben nur bei gesetztem Flag `x` das Wechseln in das Verzeichnis.

Die Zugriffsrechte für jede Datei werden jeweils vergeben für

- `u` Eigentümer des Objektes
- `g` Gruppe des Objektes
- `o` alle anderen Benutzer
- `a` alle

Die Rechte einer Datei werden mit Hilfe des Kommandos `chmod` verändert.

Kommando `chmod`

---

**Syntax:**

```
chmod [ugoa][+-=][rwx] Datei(en)
```

Ändern der Zugriffsrechte von Dateien bzw. Verzeichnisssen. Die Rechte für die einzelnen Benutzerkategorien können nach dem Schema

```
uuugggooo
rwxrwxrwx
```

mit `ls -l` angezeigt werden.

**Beispiel:**

```
wegner@arber:~/numerik > ls -l
total 4
-rw-r--r-- 1 wegner stud 100 Oct 20 15:02 hello.cpp
wegner@arber:~/numerik > chmod go+w hello.cpp
wegner@arber:~/numerik > ls -l
total 4
-rw-rw-rw- 1 wegner stud 100 Oct 20 15:02 hello.cpp
wegner@arber:~/numerik >
```

**Übung:**

1. *Nehmen Sie dem Verzeichnis* `uebung0` *das Ausführungsrecht! Können Sie nun noch dorthin wechseln?*

2. *Nehmen Sie der Datei* `linux.txt` *alle Rechte! Wer kann diesen Zustand wieder rückgängig machen?*



# 4 Dateien editieren und drucken

Zum Erzeugen bzw. Ändern von Textdateien wird ein *Editor* benötigt. Unter UNIX existiert eine Vielzahl von Editoren, die sich vor allem hinsichtlich Bedienkomfort und Speicherplatzbedarf unterscheiden.

## Der Editor `vi`

Der einzige auf `allen` UNIX-Systemen verfügbare Editor ist `vi`.

`vi` ist

- komplett tastaturgesteuert
- außerordentlich flexibel
- relativ schwierig zu erlernen

Wer unbedingt will, kann den Gebrauch dieses Editors erlernen. Eine einfachere und komfortablere Alternative, die ebenfalls auf fast allen UNIX Rechnern implementiert ist, ist

## Der Editor `emacs`

Der Editor `emacs` arbeitet in einem eigenen Fenster, das menü- und mausgesteuert ist. `emacs` erfordert einen relativ hohen Speicherplatzbedarf (bei heutigen Rechnern kein Problem), da man mit dieser Anwendungen viel mehr machen kann als nur zu editieren.

### Übung:

1. *Editieren Sie das Programm* `hello.f90`*! Starten Sie dazu* `emacs` *mit* `emacs hello.f90` *über die Kommandozeile! Versuchen Sie, die Kommentare in den Zeilen, die mit* `//` *beginnen, zu ändern!*

2. *Sichern Sie das Programm mit* `Ctrl X Ctrl S`*!*

3. *Verlassen Sie* `emacs` *mit* `Ctrl X Ctrl C`*!*



Kommando `cat`

---

### Syntax:

```
cat  Datei
```

Liest den Inhalt der angegebenen Datei und gibt deren Inhalt auf den Standardausgabekanal aus.

`cat` ist wie sehr viele andere UNIX-Kommandos ein *Filter*, der statt aus einer Datei auch von der Standardeingabe lesen kann. Daher lassen sich mit `cat` auch (kleine) Dateien direkt editieren und damit erzeugen. Die Ausgabe muß in diesem Fall mit `>` in eine Datei *umgeleitet* werden. `cat` erwartet dann eine Eingabe von der Konsole, die mit `Ctrl D` abgeschlossen wird.

### Beispiel:

```
wegner@arber:~ > cat > test
Das ist ein Test.
^D
wegner@arber:~ > cat test
Das ist ein Test.
wegner@arber:~ > more test
Das ist ein Test.
wegner@arber:~ >
```

### Übung:

1. *Betrachten Sie die Datei* `.plan`*!*

2. *Betrachten Sie nun die Datei* `linux.txt`*! Ist* `cat` *dafür geeignet?*

## Kommando `more`

### Syntax:

```
more Datei
```

`more` erlaubt das seitenweise Betrachten auch größerer Dateien. Wichtige Kommandos innerhalb von `more` sind `b` zum Zurückblättern und `q` zum Beenden.

### Beispiel:

```
wegner@arber:~ > more hello.f90
```

### Übung:

*Betrachten Sie die Datei* `linux.txt` *nun mit* `more`*! Welche Wirkung haben* `RETURN` *und die Leertaste?*



## Kommando `lpr, lpq, lprm`

### Syntax:

```
lpr -PDruckername Datei
lpq -PDruckername
lprm Jobnummer
```

`lpr` druckt eine Datei auf dem mit `Druckername` bezeichneten Drucker.

`lpq` zeigt für den mit `Druckername` bezeichneten Drucker die aktuellen Einträge in der Druckerwarteschlange jeweils mit Nummer an.

`lprm` löscht den mit `Jobnummer` bezeichneten Druckjob aus der Warteschlange.

### Beispiel:

```
wegner@arber:~ > a2ps hello.f90 -o hello.ps
[hello.f90 (Fortran): 1 page on 1 sheet]
[Total: 1 page on 1 sheet] saved into the file ‘hello.ps’
wegner@arber:~ > ls
hello*     hello.f90  nsmail/   test
hello.cpp  hello.ps   numerik/
wegner@arber:~ > lpr -Plp0 hello.ps
wegner@arber:~ >
```

### Übung:

*Drucken Sie die Datei* `linux.txt` *aus!*

## Weitere wichtige (UNIX)-Kommandos

`a2ps` Wandelt ASCII-Text in PostScript um. Oftmals notwendig, um unter Linux Text zu drucken.

```
a2ps [Optionen] textfile
-1, -2, ..., -9 vordefinierte Fontgroessen und
                Seitenlayouts. Bsp: -2: jeweils
                zwei Seiten werden auf einer angezeigt
-o              output-file (*.ps)
-P NAME         output nach Drucker NAME schicken
```

`diff` Vergleicht 2 Dateien. Bei Gleichheit *keine* Ausgabe.

`touch` Setzt den Zeitstempel einer Datei. Kann auch zum Anlegen einer (leeren) Datei benutzt werden.

`finger` Ermittelt zusätzliche Informationen zu einem Benutzernamen (Klarname, Projekt usw.).

`gv datei.ps` Zeigt PostScript-Dateien (und ähnliche Formate, z.B. `*.eps, *.pdf`) an.

`acroread datei.pdf` Zeigt Pdf-Dateien an und erlaubt einfache Manipulationen (Text und Bilder in Zwischenablage kopieren).

`gimp` (Datei). Ruft Bildverarbeitungsprogramm `gimp` auf. Erlaubt das Betrachten, Manipulieren und Drucken von Bilddateien (z.B. `*.jpg, *.tif, *.png`).



`ps2pdf13 datei.ps` Konvertiert nach Pdf. Datei `datei.pdf` wird erzeugt.

`gzip` Datei. Komprimiert Datei mittels Lempel-Ziv Algorithmus. Das file Datei.gz wird erzeugt und file Datei gelöscht. Max. Kompressionsfaktor $\sim$3.

`gunzip` Datei.gz. Dekompression.

`tar` "tape archive". Wird heutzutage hauptsächlich dazu verwendet, aus Verzeichnisbaum eine einzige Datei zu erzeugen, die dann z.B. mit email verschickt werden kann, bzw. diesen wiederherzustellen.

```
tar -cvf direc.tar direc
        erzeugt(c) Datei(f) direc.tar aus
        Verzeichnis direc. Inhalt wird
        protokolliert(v).
tar -xvf direc.tar
        stellt urspruenglichen Verzeichnisbaum
        unter originalem Namen (./direc) wieder
        her (sofern Datei direc.tar ``richtig''
        erzeugt wurde).
tar -zcvf direc.tgz direc
tar -zxvf direc.tgz
        Zusaetzliche Kompression/dekompression
```

Achtung: Befehl ist sehr "mächtig". Lesen Sie entweder die man-page oder benutzen Sie Kommando so wie angegeben.

`locate` Suchmuster. Listet Dateien in der lokalen Datenbasis auf, die Suchmuster entsprechen. Hervorragend zum Suchen von Dateien geeignet (sofern Datenbasis immer aktualisiert wird → Systemadministrator)

`find` Sucht rekursiv nach Dateien (entsprechend Suchmuster) im angegebenen Pfad.
Bsp.: `find . -name ''*.txt''` sucht im aktuellen Verzeichnis rekursiv nach allen `*.txt` Dateien.

`grep` Sucht nach Text *innerhalb* aller angegebener Dateien.
Bsp.: grep "test" ../*.txt sucht nach dem Text `test` in allen `*.txt` Dateien des übergeordneten Verzeichnisses.

# 5 Fortran 90: Ein Überblick

## Historisches

- eingeführt 1954
- Fortran II, Fortran IV, Fortran66, Fortran77
- Fortran 90 seit 1991, ANSI- und CEN-Standard

## Merkmale

- Fortran war *und ist* die meistbenutzte Sprache für naturwissenschaftliche Problemlösungen, insbesondere für Simulationen.
- Durch die Einführung von F90/95 konnten fast alle features von C/C++ nachempfunden werden, bis auf die "Hardwarenähe".
- Der Sprachumfang von F90 ist sehr groß, es existiert eine Vielzahl, der Sprache zugehörige Funktionen und Operationen. Vektor- und Matrixoperationen gehören zum Standard. Beispiel

  $$a = b * c$$

  kann Skalar-, Vektor- oder Matrix-Multiplikation bedeuten, je nach Definition von a,b,c

- Es existieren umfangreiche Programmbibliotheken, insbesondere für Lineare Algebra und Eigenwertprobleme (sowohl Quellen als auch extrem schnelle Binärobjekte), z.B. LAPACK, EISPACK.

- Die Optimierungsmöglichkeiten sind umfangreich, optimierte Programme laufen bis zu einem Faktor 10 schneller als in C++

- Parallelisierung ist mittels HPF (high performance Fortran) auf einfachste Weise moeglich, derzeit kein Analogon unter C++

- Die Grundstrukturen sind sehr einfach, und die Sprache ist (zumindest in ihrer Basiskonzeption) erheblich einfacher zu erlernen als C++.

- Wer in der Physik bleiben möchte und nicht an reiner Theorie oder reiner Experimentalphysik interessiert ist, kommt um Fortran kaum herum.

  Beispiele für physikalische Disziplinen, die fast ausschliesslich Fortran verwenden, sind: Aero-/Hydrodynamik, Astrophysik, Atomphysik, Geophysik, Kernpyhsik, Meterologie

- Fortranprogrammierung ist schnell!



## Literatur


- Online-Tutorial der Univ. Liverpool
  `http://www.liv.ac.uk/HPC/HTMLFrontPageF90.html`

- Referenzhandbuch W. Gehrke, Fortran90 Referenz-Handbuch, 1991, Hanser, München,
  ISBN 3446163212

- **Lehrbücher**

- **Rabenstein,D.**: *Fortran 90. Lehrbuch.*
  1995, (Taschenbuch), Hanser, München,
  ISBN 3446182357

  **Heisterkamp,M./Rotthäuser,K.-H.**: *FORTRAN 90.*
  1992, (Taschenbuch), Spektrum Akad. Vlg.,
  ISBN 3860255312

  **Metcalf,M./Reid,J.K.**: *Fortran 90/95 Explained.*
  1999, (Taschenbuch),Oxford Univ. Pr.,
  ISBN 0198505582


## Internetresourcen

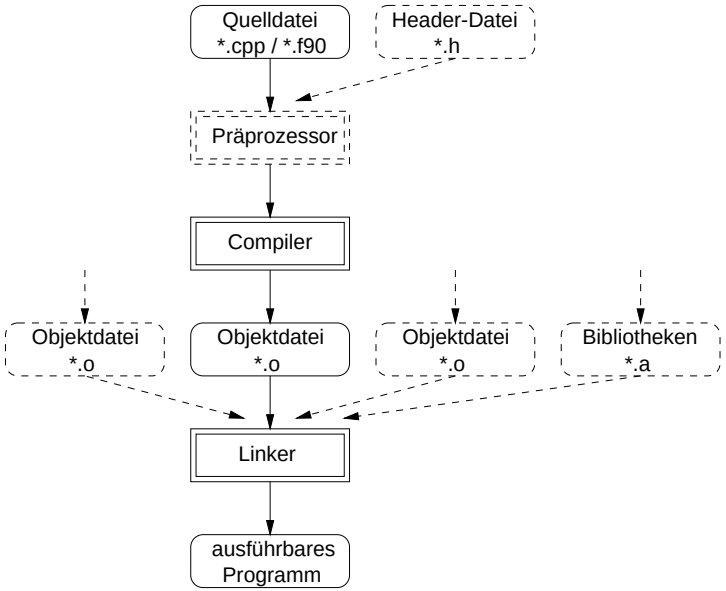
- Metcalf's Fortran Information
  `http://www.fortran.com/metcalf`

- Michel Olagnon's Fortran 90 List
  `http://www.fortran-2000.com/MichelList/`

# 6 Erstellung lauffähiger Programme

## Phasen der Programmgenerierung

Bei der Erstellung eines lauffähigen C++ - bzw. Fortran-Programms werden immer die gleichen Schritte durchlaufen:




Der *Präprozessor* ersetzt symbolische Konstanten und Makros und fügt zusätzliche Dateien ein. Entfällt normalerweise für Fortran-Programme.

Der *Compiler* übersetzt die aktuelle Datei in Maschinencode.

Der *Linker* bindet alle Objektdateien und evtl. Bibliotheken zum lauffähigen Programm.



Kommando `g++`

---

**Syntax:**

```
g++ [-c] [-g] [-O] datei.cpp [-o outputfile]
```

Im Rahmen der Vorlesung wird der GNU-C++-Compiler eingesetzt, der zu praktisch jeder Linux-Distribution dazugehört. Der Linker ist bereits integriert, kann aber separat aufgerufen werden. Wichtige Optionen:

`-o` Name der ausführbaren Datei kann angegeben werden, sonst ‘‘`a.out`’’.

`-c` Nur compilieren, nicht linken.

`-g` Debug-Information mit aufnehmen. Zur späteren Fehlersuche mit Debugger notwendig, macht das Programm aber größer und langsamer.

`-O` Optimierung.

**Beispiel:**

```
wegner@arber:~ > g++ hello.cpp
wegner@arber:~ > ./a.out
Hello, world!
wegner@arber:~ > g++ hello.cpp -o hello
wegner@arber:~ > ./hello
Hello, world!
```

Die Zeichen `./` vor `a.out` bzw. `hello` sind meistens erforderlich, um der `shell` mitzuteilen, dass diese ausführbaren Programme im *aktuellen* Verzeichnis vorhanden sind.

Nun in zwei Schritten: erst compilieren, dann linken

**Beispiel:**

```
wegner@arber:~ > g++ -c hello.cpp
wegner@arber:~ > g++ hello.o -o hello
```

Kommando `ifort`

---

**Syntax:**

```
ifort [-c -g -O] datei.f90 [-o outputfile]
```

Im Rahmen des numerischen Praktikums wird der Intel Fortran 95-Compiler eingesetzt. Auch hier ist der Linker bereits mit enthalten, kann aber ebenfalls separat aufgerufen werden. Falls keine Möglichkeit zur Verwendung von `ifort` besteht, kann auch der GNU Fortran 95-Compiler `gfortran` (mit den gleichen oder ähnlichen Optionen) verwendet werden. `gfortran` steht auch als Windows-binary kostenlos zur Verfügung.

Wichtige Optionen:

`-o` Name der ausführbaren Datei kann angegeben werden, sonst ‘‘`a.out`’’.

`-c` Nur compilieren, nicht linken.

`-g` Debug-Information mit aufnehmen.

`-O` Optimierung.

**Beispiel:**

```
wegner@arber:~ > ifort hello.f90 -o hello
wegner@arber:~ > ./hello
Hello, world!
wegner@arber:~ >
```

**Übung:**

1. *Übersetzen Sie Ihr Fortran - HelloWorld und starten Sie das Programm!*

2. *Übersetzen Sie Ihr Programm nun mit der Option* `-c` *und linken Sie in einem zweiten Schritt durch den Aufruf von* `ifort hello.o`*!*



## Fehlersuche

Nach erfogreichem Compilieren und Linken kann ein Programm durch Aufruf über seinen Namen (`a.out` oder der mit `-o` selbstgewählte Name) gestartet werden. Das muß aber nicht heißen, daß es

1. auch erfolgreich bis zu Ende läuft und

2. das Programm sich inhaltlich im Sinne des Programmierers verhält.

Für die dann notwendige *Fehlersuche* ist normalerweise ein *Debugger* erforderlich. Dieser erlaubt es

- die einzelnen Anweisungen schrittweise zu durchlaufen

- dabei die Werte der Variablen zu verfolgen und ggf. zu ändern

- das Programm an definierten Punkten (*Breakpoints*) anzuhalten.

Ein unter Linux frei verfügbarer und fensterorientierter Debugger ist `DDD` (*Data Display Debugger*), dessen aktuelle Version u.a für den C++ und Fortran (mit der Option `-gdb`) Compiler arbeitet.

Der Aufruf erfolgt einfach über

```
ddd (-gdb) Programmname
```

wobei Programmname der Name des *ausführbaren* Programmes (z.B. `a.out`) ist.

Für F90-Programme verwendet man alternativ den von Intel mitgelieferten debugger `idb`, der sowohl zeilenorientiert also auch über eine GUI (Aufruf dann: `idb -gui`) arbeitet.

```
idb Programmname
```

Um ein Programm debuggen zu können, muß beim Compilieren (mit der Option `-g` die *Debug-Information* mit in das ausführbare Programm aufgenommen werden.

**Beispiel:**

```
wegner@arber:~ > ifort -g hello.f90 -o fhello
wegner@arber:~ > idb fhello
Linux Application Debugger for 32-bit applications,...
------------------
object file name: fhello
Reading symbolic information from fhello...done

(idb)
(idb) run
 Hello, world!
 Program has exited with status 0.
(idb)
(idb) quit
 wegner@arber:~ >
```

**Anmerkung:** zur Zeit arbeitet der `idb` debugger aufgrund von Inkonsistenzen mit dem aktuellen Betriebssystem nicht. Verwenden Sie deshalb den `ddd`.

**Übung:**

1. *Laden Sie das Programm pi.f90 von der Homepage des Praktikums herunter. Übersetzen Sie das Programm mit der Option* `-g`*!*

2. *Starten Sie* `ddd -gdb` *für Ihr ausführbares Programm! Machen Sie sich mit den wichtigsten Menüpunkten (*`Run`, `Step`, `Next`, `...`*) und Buttons (*`Break`, `Print`*) in der Steuerelementleiste vertraut.*

# 7 UNIX-Shells

Die *Shell* (Schale) ist ein Dienstprogramm, über das der Benutzer mit dem Betriebssystem kommuniziert und das die Interpretation eingegebener Kommandos übernimmt.

## Verschiedene UNIX-Shells

Da die Shell nicht direkt zum Betriebssystemkern gehört, haben sich im Lauf der Zeit eine Reihe verschiedener Versionen etabliert:

- *Bourne-Shell (sh)*. Die bekannteste und am weitesten verbreitete Shell, benannt nach Erfinder Steven Bourne. Unter Linux ist die *bash, Bourne again shell* verbreitet.

- *C-Shell (csh)*. Wurde in Berkeley entwickelt und orientiert sich an der Programmiersprache C. Eine Erweiterung der C-Shell ist die *tcsh*.

- *Bash-Shell (bash)*. Erweiterung der Bourne-Shell und auf vielen Systemen die Standard-Shell.

Jede Shell kennt einen Satz von *Systemvariablen*, die durch benutzerdefinierte Variablen ergänzt werden und die für in der Shell laufende Programme die *Prozeßumgebung* darstellen können.

Darüberhinaus läßt sich die Shell auch als *Programmiersprache* einsetzen.



## Shell-Skripte

*Shell-Skripte* sind kleine Programme, die aus UNIX-Kommandos und Shell-internen Programmflußkonstrukten (Verzweigungen, Schleifen etc.) bestehen und sich äußerlich wie eingebaute UNIX-Kommandos verhalten, trotzdem aber in Text- und nicht in Binärform vorliegen. Sie werden von der Shell *interpretiert*.

Die beim Schreiben von Shell-Skripten zu beachtende Syntax differiert zwischen den einzelnen Shells zum Teil beträchtlich.

Einige Shell-Skripte werden in bestimmten Situationen *automatisch* aufgerufen:

- `.login` bzw. `.profile` werden, falls vorhanden, immer beim Anmelden, also für die *Login-Shell*, genau einmal ausgeführt.

- `.bashrc` und `.cshrc` bzw. `.tcshrc` werden immer aufgerufen, wenn eine neue `bash` oder `csh/tcsh` geöffnet wird.

**Übung:**

1. *Kopieren Sie sich die Datei* `.tcshrc` *aus* `ubung0` *in Ihr Home-Verzeichnis und sehen Sie sich die Datei an!*

2. *Eröffnen Sie jetzt eine* tcsh *durch Eingabe von* `tcsh` *in Ihrem aktuellen Terminal! Was passiert? Verlassen Sie die* tcsh *wieder mit* `exit`*!*

## Ein-/Ausgabeumlenkung

Alle UNIX-Kommandos benutzen *Ein- und Ausgabekanäle*, um Daten zu lesen und Daten auszugeben. Das sind normalerweise die Tastatur und der dem jeweiligen Benutzer zugeordnete Bildschirm.

Diese Standardkanäle können in der Shell für jeweils ein Kommando so umgeleitet werden, daß das Kommando entweder direkt aus einer Datei statt von der Tastatur liest oder in eine Datei statt auf den Bildschirm schreibt. Zur Umleitung wird dann entweder das Zeichen '>' (Ausgabe) oder '<' (Eingabe) benutzt.

Mit '>>' wird die Ausgabe an eine existierende Datei angehängt.

**Beispiel:**

```
wegner@arber:~ > ls
hello.cpp  linux.txt  numerik/
hello.f90  nsmail/
wegner@arber:~ > cat linux.txt > linux2.txt
wegner@arber:~ > ls
hello.cpp  linux.txt   nsmail/
hello.f90  linux2.txt  numerik/
```



## Pipes

Viele UNIX-Kommandos sind darüberhinaus sogenannte *Filter*: Sie lesen von der Standard-Eingabe und schreiben auf die Standard-Ausgabe. Deshalb können Sie durch sogenannte *Pipes* (Röhren) so verkettet werden, daß die Ausgabe eines Kommandos als Eingabe eines anderen Kommandos fungiert:


Zwischen die beiden Kommandos wird dann ein '|' gesetzt.

Eine Ausgabeumlenkung in eine Datei mit '>' oder '>>' kann immer nur *am Ende* einer solchen Verkettung stehen.

**Beispiel:**

```
wegner@arber:~ > man g++ | a2ps -P printer
[Total: 151 pages on 76 sheets]
wegner@arber:~ >
```

Hier wird die Beschreibung von `g++` aus dem Online-Handbuch direkt formatiert und auf dem Drucker `printer` gedruckt.

# 8 Prozeßverwaltung

Ein *Prozeß* ist ein *laufendes Programm* und besteht aus

- dem Programm selbst sowie der
- Umgebung des Programms, die aus allen zum korrekten Ablauf des Programms erforderlichen Zusatzinformationen besteht.

*Kenndaten* eines Prozesses sind u.a.

- eine eindeutige Prozeßnummer (PID),
- Nummer des *Elternprozesses*,
- Benutzer- und Gruppennummer des *Besitzers* und
- die *Priorität* des Prozesses.

Wird ein Prozeß aus der Shell gestartet, steht diese für weitere Eingaben bis zum Ende des Prozesses nicht mehr zur Verfügung. Deshalb können Prozesse bzw. Programme im *Hintergrund* ablaufen. Die Kommandozeile muß beim Aufruf des entsprechenden Kommandos dann mit einem '`&`' abgeschlossen werden.

**Beispiel:**

```
wegner@arber:~ > firefox &
[1] 21749
wegner@arber:~ >
```

**Übung:**

*Starten Sie das Programm* `xeyes` *(verfolgt Ihren Mauszeiger mit Augenbewegungen) im Hintergrund!*



Kommando `ps`

---

**Syntax:**

```
ps [-al] [-u user]
```

Zeigt laufende Prozesse mit ihren Kenndaten an. Ohne zusätzliche Optionen werden die in der aktuellen Shell laufenden (eigenen) Prozesse angezeigt. Wichtige Optionen:

`-a` Zeigt alle Prozesse an, die irgendeinem Terminal (tty) zugeordnet sind.

`-l` Anzeige im Langformat. Zusätzliche Informationen über Eigentümer, Elternprozeß usw.

`-u` Erlaubt die selektive Anzeige der Prozesse, die einem bestimmten `user` gehören.

**Beispiel:**

```
wegner@arber:~ > ps
  PID TTY          TIME CMD
21733 pts/4    00:00:00 bash
22197 pts/4    00:00:00 xterm
22198 pts/5    00:00:00 bash
22212 pts/4    00:00:00 ps
wegner@arber:~ >
```

**Übung:**

*Lassen Sie sich alle Prozesse anzeigen, die momentan in Ihrer Shell laufen!*

Kommando `kill`

---

## Syntax:

```
kill [-9] PID
```

Beendet den Prozeß mit der Nummer `PID`. Kann nur vom Eigentümer des Prozesses oder von *root* ausgeführt werden.

Wichtige Option:

`-9` für "hartnäckige Fälle", die sich mit normalem `kill` nicht beenden lassen.

## Beispiel:

```
  PID TTY          TIME CMD
21733 pts/4    00:00:00 bash
22197 pts/4    00:00:00 xterm
22198 pts/5    00:00:00 bash
22212 pts/4    00:00:00 ps
wegner@arber:~ > kill 22197
wegner@arber:~ > ps
  PID TTY          TIME CMD
21733 pts/4    00:00:00 bash
22214 pts/4    00:00:00 ps
[1]+  Exit 15  xterm
wegner@arber:~ >
```

## Übung:

*Beenden Sie* `xeyes` *über* `kill`*!*