

# Einführung in Unix

Edouard Lamboray  
Informatik I für D-ITET (WS 03/04)

- Rolle des Betriebssystems
- Unix Prozesse
- Unix Dateisystem
- Die wichtigsten Unix Kommandos
- Programmieren und Programmierwerkzeuge



Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich

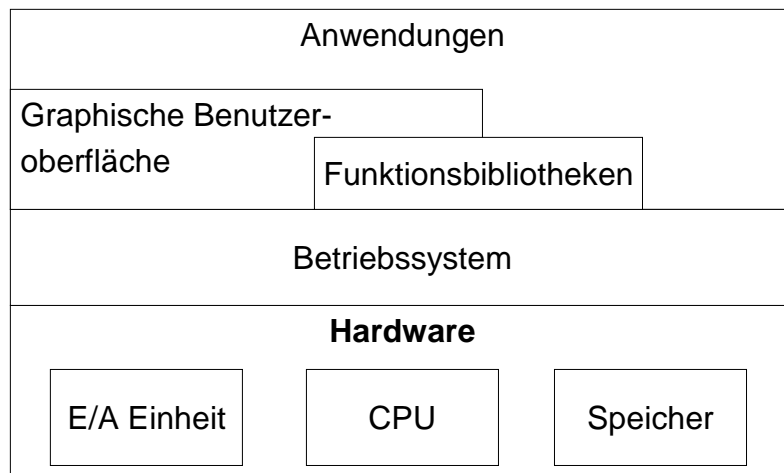
Copyright: E. Lamboray, ETHZ, 2003

2



Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich

## Der Computer - Systemumgebung



## Betriebssystem

- Abstraktion
  - ◆ Kapseln der technischen Details der Hardware
- Schnittstelle für Anwendungsprogramme
  - ◆ Application Programming Interface (API)
- Koordination und Zuteilung von Betriebsmitteln
  - ◆ Prozessorkapazität, Hauptspeicher, Massenspeicher, Zugriff auf E/A Einheiten
  - ◆ Ablaufplanung (scheduling) des Mehrprozessbetriebs (multitasking)
  - ◆ Multitasking  $\neq$  Multiprozessor

## Betriebssystem

- Schutz mehrerer gleichzeitig oder nacheinander aktiver Benutzer (multiuser)
- Bedienschnittstelle für Benutzer
  - ◆ Betriebssystemabhängig: Microsoft Windows
  - ◆ Betriebssystemunabhängig: X-Windows
- Beispiele:
  - ◆ Microsoft Windows
  - ◆ MacOS, PalmOS, VxWorks
  - ◆ Unix: BSD, HPUX, Solaris, Irix, Linux

## Unix

---

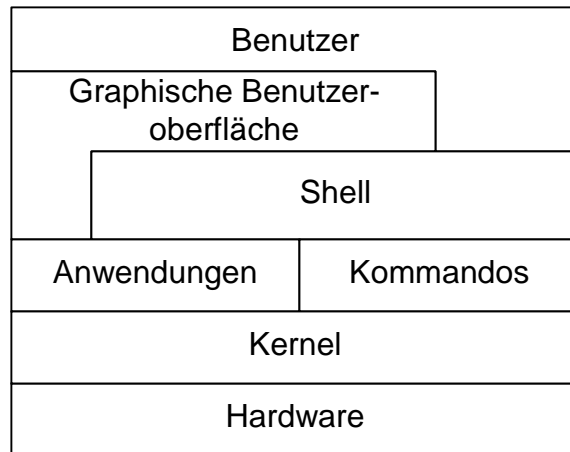
- Multitasking Betriebssystem
  - ◆ Prozessorientiert
  - ◆ CPU Scheduling basiert auf Prioritäten
- Mehrstufiger Baum als Dateisystem
- Dateien und E/A Einheiten werden ähnlich behandelt
- Multiuser und Multiprozessor möglich
- Kapselung der Maschinenarchitektur
  - ◆ Erleichtert die Portabilität von Programmen

## Unix

---

- Open-Source und proprietäre Implementierungen
  - ◆ BSD, Linux
  - ◆ AIX (IBM), HP-UX (Hewlett-Packard), Irix (sgi), SunOS/Solaris (Sun Microsystems)
- Unix-Betriebssystem existiert für fast alle Hardware-Plattformen
- Alle gängigen Compiler sind für Unix verfügbar
- Viele verschiedene Anwendungen
  - ◆ <http://www.gnu.org>

## Schichtenmodell



## Historie

- 1969: Erste Unix-Version von Ken Thompson und Dennis Ritchie, Bell Labs in Murray Hill, NJ
- Multics: Vorläufer von Unix
- Programmiersprache C wurde zur Unterstützung von Unix entwickelt
- 1973: Unix Kernel in C implementiert
- 1976: Version 6 wird einer grossen Benutzergruppe, auch ausserhalb von AT&T, zugänglich gemacht
- 1978: Erstes Berkeley Unix für VAX
  - ◆ BSD: Berkeley Software Distribution
- 1994: Linux 1.0

## Standards

- Posix (Portable Operating System Interface for Computer Environments)
  - ◆ IEEE, ANSI
- Motif window environment

## Anmelden

- Anmelden (log in) mittels
  - ◆ Benutzername, Beispiel: `lamboray`
  - ◆ Passwort, Beispiel: `as%K17u`
- Shell / Kommandointerpreter / Terminal
  - ◆ Bourne shell: `sh`
  - ◆ C shell: `csh`
  - ◆ C shell with file name completion and command line editing:  
`tcsh`
  - ◆ Korn shell: `ksh`
  - ◆ „Bourne again shell“: `bash`
- Ändern des Passwortes: `passwd`

## Unix Kommandos

- `command [options] [attributes]`
- Optionen: Kontrolle und Konfiguration des Kommandos
  - ◆ Beginnen mit -
- Attribute: Spezifizieren von Zusatzdaten, wie z.B. Dateinamen
- Beispiele:
  - > `ls -al`
  - ◆ Zeigt alle Dateien im aktuellen Verzeichnis an
  - > `ls -a -l *.h`
  - ◆ Zeigt alle Dateien im aktuellen Verzeichnis an, deren Name auf `.h` endet

## Regular Expressions

- Definieren ein Textmuster, welches erfüllt werden soll
- Positionsunabhängige Sonderzeichen:
  - ein einfaches Zeichen (character)
  - \* irgendeine Zeichenfolge (arbitrary string)
  - [ ] Inhalt definiert eine Zeichenmenge (set of characters)
  - \ setzt das folgende Zeichen zwischen Anführungszeichen
- `grep [options] pattern [file]` (global reg. expr. print)
- `fgrep` und `egrep` (fast / extended grep)
- Beispiele:
  - ◆ > `ls *.*[0-9].*`
  - ◆ > `cat myfile | grep \*`
  - ◆ In einer Datei: ... Zeilen- \
  - `umbruch`

## Prozesse

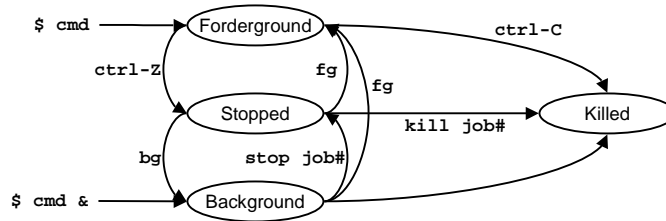
- Prozess: Ablauf eines sequentiellen Programs
- Parent und Child Prozesse
- Daemons: Hintergrundprozesse, welche kontinuierliche Aufgaben erledigen
  - ◆ Beispiel: E-Mail, Drucken
- Pipe | als Kommunikationskanal zwischen zwei Prozessen:
  - ◆ Output von Prozess 1 als Input zu Prozess 2
  - ◆ Beispiel: `> ls -al | grep .h`

## Prozessmodell

- Das gleiche Programm kann in verschiedenen Prozessen gleichzeitig ausgeführt werden.
- Graphische Benutzerschnittstellen, wie z.B. X-Windows, stellen mehrere Pseudo-Terminals zur Verfügung
- In jedem dieser Terminals können ein oder mehrere Prozesse gestartet werden
- Ereignisse werden den Prozessen über Signale kommuniziert.
- Jeder Prozess liefert einen Rückgabewert:
  - ◆ `== 0`: okay
  - ◆ `!= 0`: abnormale Terminierung

## Vorder-/Hintergrundprozess

- Shell interpretiert Kommandos und führt sie in Child-Prozessen aus
  - ♦ Synchron/Asynchron oder Vordergrund/Hintergrund
  - ♦ Hin- und Herschalten: `ctrl-Z` und `bg / fg`
  - ♦ Beispiel: `> xxx` versus `> xxx &`



- Unterbrechen des aktuellen Vordergrundprozesses: `ctrl-C`

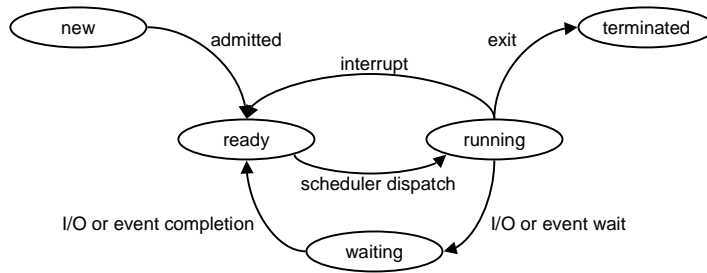
## Prozesskommandos

- Anzeigen der aktuellen Prozesse:  
`ps` oder `ps -ef` oder `ps -fu username`  
(process status)
- Anzeigen der Prozesse, geordnet nach Ressourcennutzung:  
`top`
- Prozess unterbrechen:  
`kill pid` oder  
`kill -9 pid`

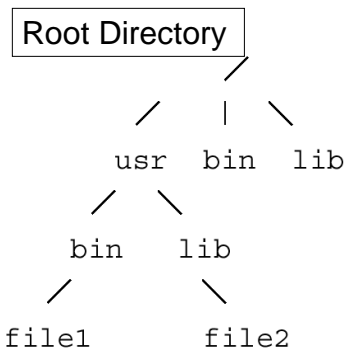


## Prozesszustände

- Erstellen, Terminieren
- Rechnend, Rechenbereit, Wartend



## Dateisystem am Beispiel



- Drei Dateitypen:
    - ◆ Ordentliche Dateien (Files) enthalten Daten
    - ◆ Spezialdateien, erlauben z.B. Zugriff auf I/O Einheiten
    - ◆ Verzeichnisse (Directories) gruppieren eine Menge von Dateien
      - Aktuelles Verzeichnis: .
      - Väterverzeichnis: ..
- Jede Datei ist durch einen absoluten Pfad definiert:  
**/usr/lib/file2**
- Relative Pfade: ab **/usr/lib/file2** nach **../bin/file1**

## Dateimanipulation

- Jeder Benutzer hat ein eigenes Homedirectory: `~`
- Homedirectory eines bestimmten Benutzers:  
`~lamboray`
- Neues Verzeichnis erstellen:  
`> mkdir dirname` (make directory)
- Bewegen durch die Dateistruktur:  
`> cd ../myDir/anotherDir/` (change directory)
- Aktuelles Verzeichnis:  
`> pwd` (print working directory)
- Versteckte Dateien, Beispiel: `.myfile`
- Gross- und Kleinschreibung beachten:  
`MyFile != myfile`

## Dateimanipulation

- Dateien löschen: `> rm filename`
- Empfehlung: `> rm -i filename`
  - Bestätigung vor dem definitiven Löschen verlangt
- Ordner löschen: `> rmdir directory`
  - Ordner muss leer sein
- Verweis (Link) auf eine Datei
  - Keine Kopie der Datei!
  - Hardlink: speichert die interne Adresse der Datei
  - Softlink: speichert den Pfadnamen der Datei
  - `> ln -s ../myDir/myFile alsoMyFile`
- Tipps:
  - Automatisches Erweitern des Dateinamens: **TAB**
  - History: **↑** und **↓**

Copyright: E. Lamboray, ETHZ, 2003 21

Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich

## Zugriffsrechte

---

- 3 Grundrechte:
  - ◆ Lesen (read)
  - ◆ Schreiben (write)
  - ◆ Ausführen (execute)
- 3 Benutzergruppen
  - ◆ Besitzer (owner)
  - ◆ Gruppe (group)
  - ◆ Alle anderen (others)
- Gilt für Dateien und Ordner

Rechte	Owner	Group	Dateigrösse, Datum	Name
--------	-------	-------	--------------------	------

```

[edy@tiepolo-vmware SomeFiles]$ ls -l
total 20
-rw-r--r--  1 edy   edy    24 Oct 19 20:05 file1.txt
-rw-rw-r--  1 edy   edy    37 Oct 19 20:09 file2.txt
lrwxrwxrwx  1 edy   edy     9 Oct 19 20:06 file3.txt -> file2.txt
drwxrwxr-x  2 edy   edy  4096 Oct 19 20:06 mydirectory
-rwx-----  1 edy   edy     8 Oct 19 20:08 showFile
-rwxr-xr-x  1 edy   edy     8 Oct 19 20:08 showFileToEverybody
[edy@tiepolo-vmware SomeFiles]$
```

Copyright: E. Lamboray, ETHZ, 2003 22

Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich

## Festlegen der Zugriffsrechte

---

- `chmod modespecs filename` (change mode)
- Liste von Zugriffsveränderungen, durch Kommata getrennt
 

<b>u</b> Besitzer	+ Hinzufügen	<b>r</b> Lesen
<b>g</b> Gruppe	- Löschen	<b>w</b> Schreiben
<b>o</b> Alle ändern	= Genau diese Rechte	<b>x</b> Ausführen
- **a** Alle (ugo)
- Oktalzahl: ugo mit jeweils **4\*r+2\*w+1\*x**
- Beispiel:
  - ◆ > `chmod ug=rwx, o=rx myfile`
  - ◆ > `chmod 775 myfile`

## Umgebungsvariablen

- `setenv [variablename] [value]`
- `echo variablename`, z.B.: `> echo $HOME`
  - ◆ `$` bewirkt Auswertung der Variable
- Variablenbeispiele:
 

<code>PATH</code>	Verzeichnisse, welche bei Aufruf nach dem entsprechenden Executable durchsucht werden
<code>LD_LIBRARY_PATH</code>	Verzeichnisse, welche bei Aufruf eines Executables nach dynamischen Bibliotheken durchsucht werden
- Beispiel: Hinzufügen des aktuellen Verzeichnisses in die `PATH` Variable:
 

```
> setenv PATH .:$PATH
```

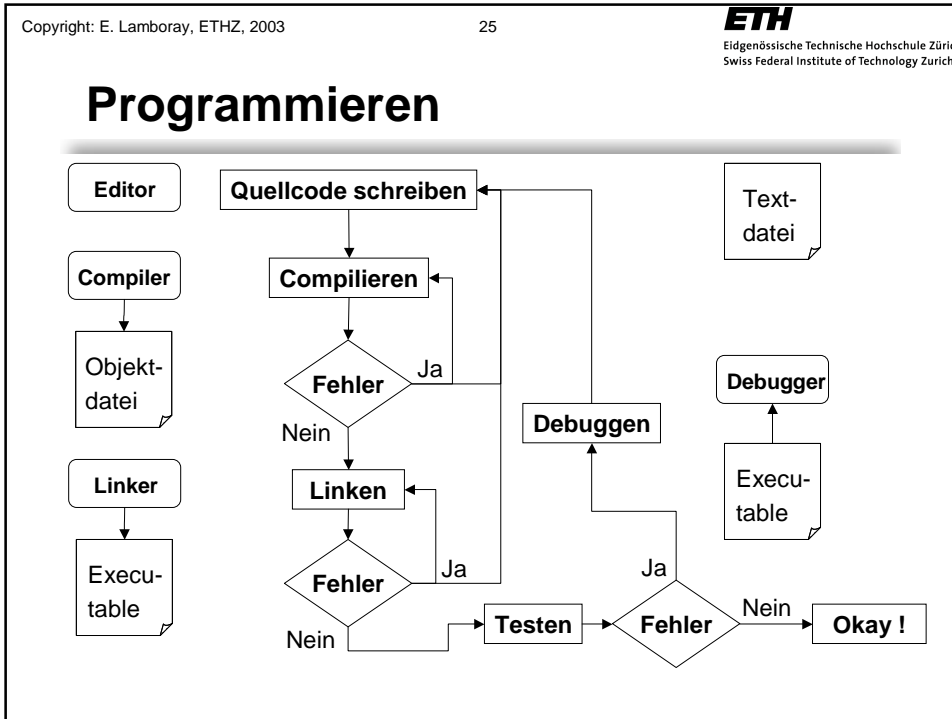
## Andere wichtige Kommandos

- Dokumentation: `man commandname`
- Suche nach entsprechendem Kommando:
 

```
apropos keyword
```
- Anzeige des absoluten Executable-Pfades:
 

```
which commandname
```
- Vergleichen von Dateien: `cmp file1 file2` oder `diff file1 file2`
- Drucken: `lp`, `cancel` und `lpstat`
- Definieren von Kürzel:
 

```
alias aliasname command
```



- Copyright: E. Lamboray, ETHZ, 2003 26 **ETH**  
Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich
- ## Programmierfehler
- **Syntaxfehler**
    - ◆ werden vom Compiler gemeldet
  - **Linkfehler**
    - ◆ auf fehlende Objektdateien oder Bibliotheken zurückzuführen
  - **Laufzeitfehler: semantische/logische Fehler**
    - ◆ werden nur durch Testen und Analysieren des lauffähigen Programms entdeckt
    - ◆ Laufzeitfehler sind „teuer“:
      - Wenig Unterstützung seitens des Rechners
      - Kosten für Benutzer des fehlerhaften Programms
    - ◆ Qualität und Umfang der Tests sind entscheidend!

## Editor – Emacs / Xemacs

- Schreiben und Verändern von Textdateien
- „Emacs is the extensible, customizable, self-documenting real-time display editor.“  
(aus dem Emacs manual)
- Kommandoeingabe in Mini-buffer
- Verwenden von Shortcuts beschleunigt das Editieren
- Vorteile von Xemacs:
  - ◆ Besseres Fontmanagement
  - ◆ Verbessertes Benutzerinterface
    - Toolbar
    - Dialogboxen
- <http://www.xemacs.org> und  
<http://www.gnu.org/software/emacs/emacs.html>

## Compiler – gcc und g++

- Compiler für C (gcc) und C++ (g++)
- Wichtige Flags:
 

<code>-g</code>	Debugging Information	<code>-o</code>	Targetname
<code>-c</code>	Compilieren & assemblieren	<code>-s</code>	Assemblercode generieren
<code>-I</code>	Include-Verzeichnisse	<code>-Wall</code>	alle Warnings anzeigen
<code>-L</code>	Bibliothek-Verzeichnisse	<code>-O</code>	Optimierung
- `-l $libraryname$`  zu linkende Bibliothek
- Compilieren:
 

```
> g++ -g -c file1.cpp -I ~/include/
```
- Linken:
 

```
> g++ -o myprog myprog.o file1.o -L ~/lib/ -lmylib
```
- <http://gcc.gnu.org>

## Debugger – gdb und ddd

- Debugger: **gdb**
- Graphische Benutzeroberfläche für **gdb**: **ddd**
- Dateien müssen mit **-g** Flag kompiliert sein
- Debugmöglichkeiten:
  - ◆ Programmablauf Schritt für Schritt verfolgen
  - ◆ Setzen von Breakpoints
  - ◆ Anschauen von Variablenwerten
  - ◆ Callstack anschauen
  - ◆ Analysieren der Coredatei
- Starten: **> ddd myprog &**
- <http://www.gnu.org/manual/ddd/>

## Programmierfehler: Tipps

- Versuchen Programmierfehler zu vermeiden durch:
  - ◆ Programmieren in kleinen Schritten
  - ◆ Möglichst vollständige Tests
- Und falls es doch passiert:
  - ◆ Debugger versus Debug-Ausgaben
  - ◆ Problem einschränken:
    - Grösstmögliches Programm, welches den Fehler nicht produziert
    - Kleinstmögliches Programm, welches den Fehler produziert
  - **Fehler liegt im Unterschied zwischen beiden Programmen!**