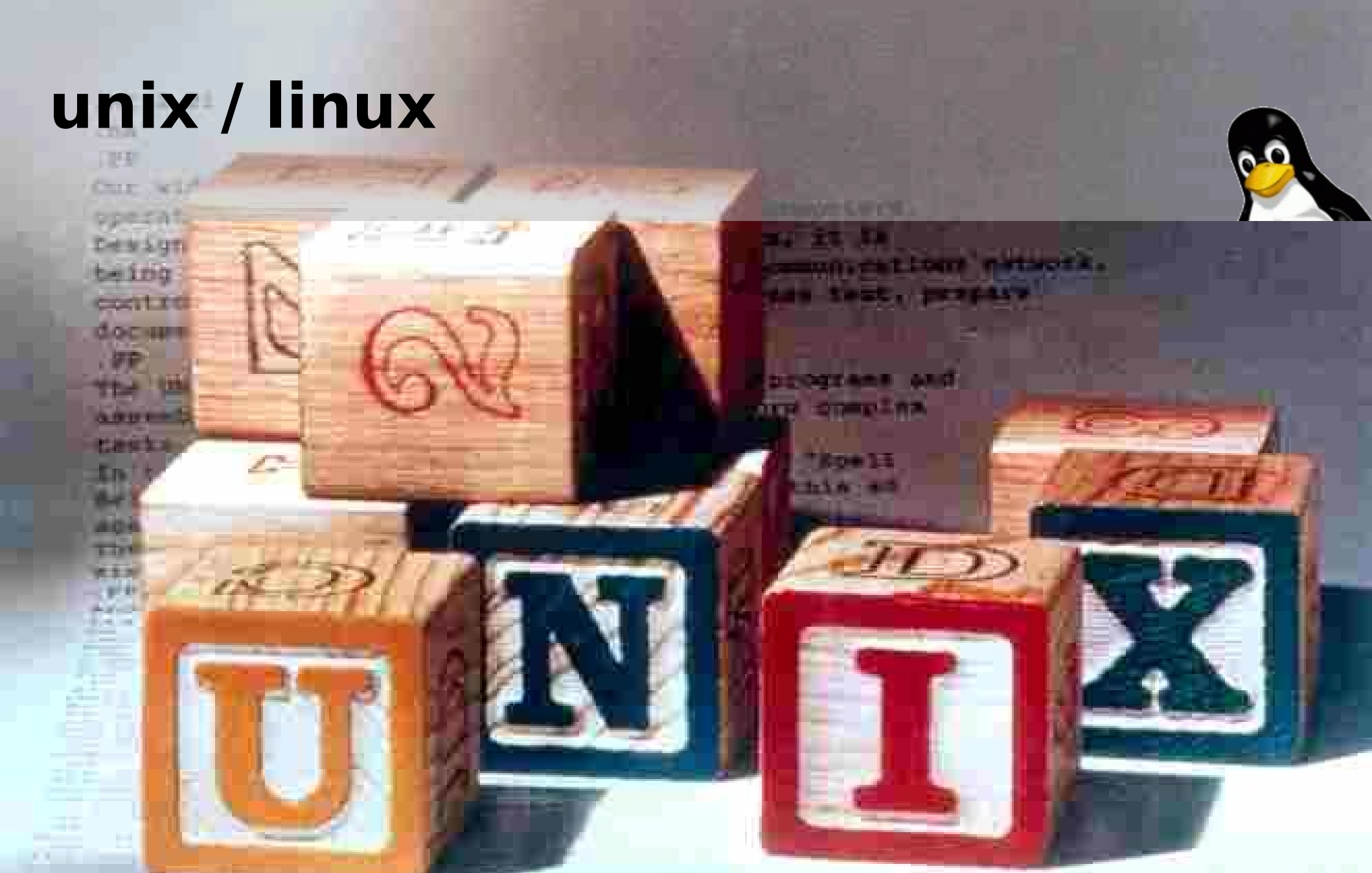
unix / linux



2 Bedienung des Unix Systems

Skript Seite 30

Bedienung des Linux-Systems

Ziele:

- Sich beim System anmelden und abmelden können
- Die Shell bedienen können
- Mit Kommandos umgehen lernen

Anmelden / Abmelden

Linux ist ein Multi-User und Tasking-System. Darum muss sich der Benutzer gegenüber dem System ausweisen.

Wenn das System weiss, welcher Benutzer da arbeitet, kann das System entscheiden ob der Benutzer das machen darf, was er machen will.

Den Anmelde-Vorgang nennt man **login**,
den Abmelde-Vorgang heisst **logout**.

Anmelden / Abmelden

Den Login Vorgang kann entweder Text orientiert oder graphisch via Display-Manager erfolgen.

Login via telnet

```
heuer@largo:~$ telnet guybrush
Trying 212.55.196.74...
Connected to guybrush.
Escape character is '^]'.
Debian GNU/Linux 4.0
guybrush login: heuer
Password:
Last login: Tue Mar 20 22:02:11 2007 from
2001:8a8:30:12:250:4ff:fe3d:b647 on pts/10
Linux guybrush 2.6.20.3 #5 SMP PREEMPT Thu Mar 15 21:35:53 CET
2007 i686

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
heuer@guybrush:~$
```

Anmelden / Abmelden

Login via ssh (Secure shell)

```
heuer@largo:~$ ssh guybrush
heuer@guybrush's password:
Linux guybrush 2.6.29 #2 SMP PREEMPT Sat Mar 28 23:06:10 CET 2009
i686

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Apr  2 11:02:51 2009 from 2a00:c38:2:1:21b:...
heuer@guybrush:~$
```

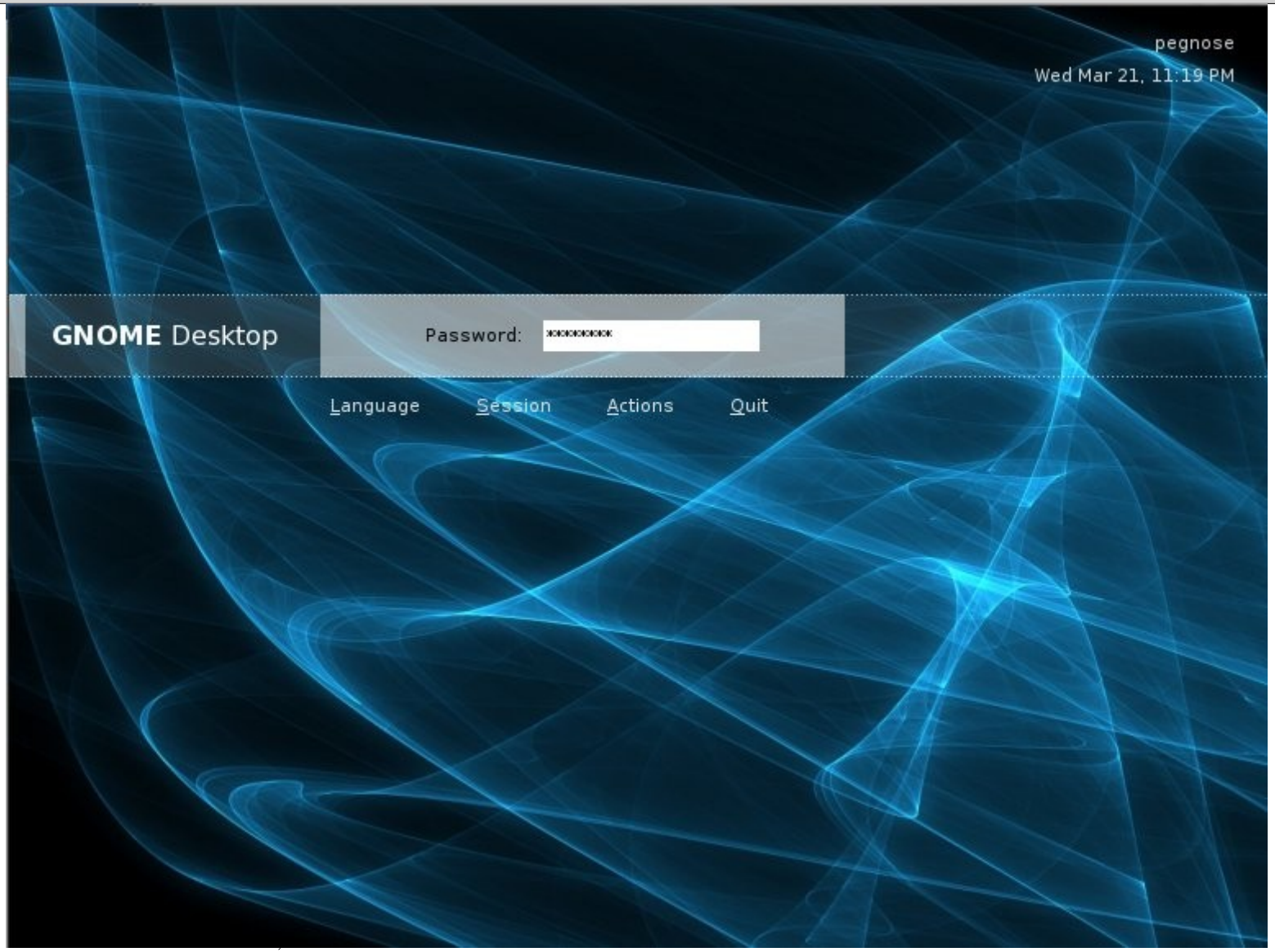
```
heuer@guybrush:~$ logout
Connection to guybrush closed.
heuer@largo:~$
```

Alternativ wird auch **ctrl-d** oder **exit** funktionieren.

Anmelden / Abmelden

Login via Display Manager

Anmelden / Abmelden



Login via Display Manager

AAA

Authentication: wer will da arbeiten
Authorization: was darf der Benutzer machen
Accounting: was hat der Benutzer gemacht

Aufgrund vom Login (Authentication) kann das System was der Benutzer machen will, erlauben [oder auch nicht] (Authorization)
Wenn das System die Aktivitäten auch noch loggt, können auf Grund dieser Daten die Resourcen verrechnet werden.

Abschalten

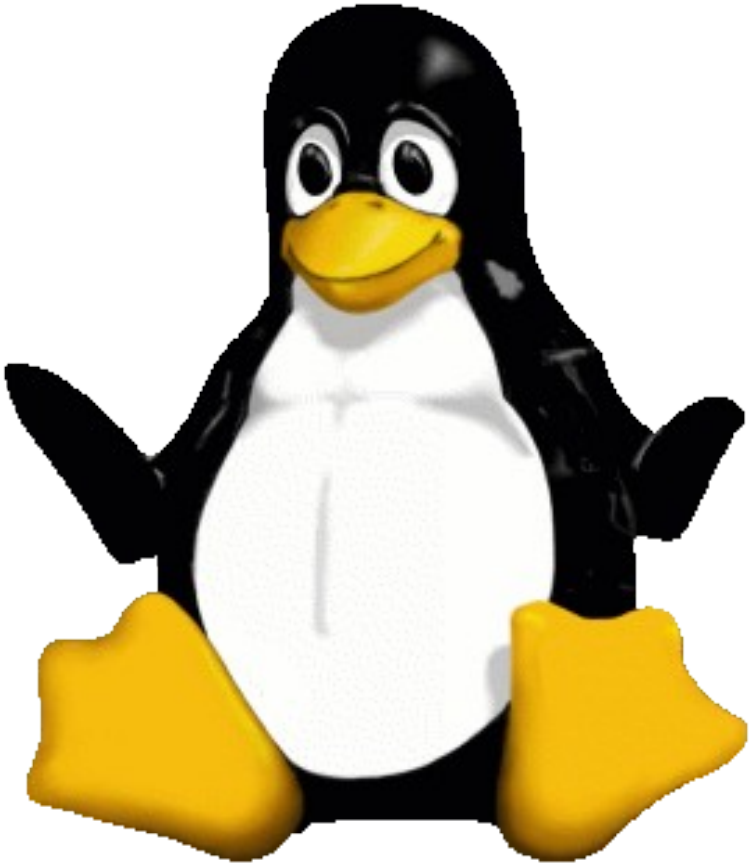
Den Rechner einschalten darf normalerweise jeder.

Einen Multitasking-Rechner abschalten ist komplizierter. Darum darf in der Regel das nur ein spezieller Benutzer (root) mit dem Befehl **`init 0`** oder **`shutdown`** tun.

Bei den graphischen Oberflächen befindet sich in der Regel ein Menüpunkt, wo sie den Rechner ausschalten können.

An der Console können sie einen Linux Rechner oft mit **`[ctrl]-[alt]-[del]`** [versehentlich] runterfahren.

Fragen ?



Anmelden / Abmelden

2.1: Versuchen Sie sich beim System anzumelden (Benutzername und Kennwort verrät Ihnen Ihr Trainer). Melden Sie sich anschliessend wieder ab.

2.2: Was passiert, wenn Sie
(a) einen nicht existierenden Benutzernamen,
(b) ein falsches Kennwort angeben? Fällt Ihnen etwas auf?
Welchen Grund könnte es dafür geben, dass das System sich so verhält, wie es sich verhält?

2.3: Prüfen Sie, ob Sie Ihr System als normaler Benutzer sauber herunterfahren dürfen, und probieren Sie es gegebenenfalls aus.

Der Systemadministrator

Normale Benutzer können mit dem System arbeiten. Sie können jedoch nur in einem speziellen – für sie reservierten Bereich (dem Home-Directory) – Dateien modifizieren. (Ok, auch im **`/tmp`** und **`/var/tmp`** kann jeder Benutzer schreiben).

Die System-Dateien können ausschliesslich vom Superuser **root** modifiziert werden. Dieser Superuser – System-Administrator – kann bei unsachgemässen Befehlen das System zerstören.
Daher ist – wenn man als Superuser arbeitet – grösste Vorsicht geboten!

Verhaltensregeln

Bei graphischen Logins verwenden sie **NIE** **root** als Login-Name (Oft erlauben die Display Manager das auch nicht).

Wenn sie mehr Rechte benötigen um einen Task zu erledigen, so wechseln sie für diesen Task zum Superuser und kehren danach sofort zum normalen Benutzer zurück.

Benutzer temporär wechseln

Mit dem Befehl **su <benutzername>** können sie zu jedem beliebigen Benutzername wechseln [vorausgesetzt sie kennen das passende Passwort!].
Wenn sie keinen Benutzername angeben wird angenommen, dass sie zum User **root** wechseln wollen.

su → switch user

Beispiel:

```
$ /bin/su
Password:
#
```

Beachten sie die unterschiedlichen Prompts: als normaler Benutzer endet der Prompt mit **$** als SuperUser **#**

Benutzer temporär wechseln

Mit dem Befehl **sudo <befehl> <optionen> <argumente>** können einzelne Befehle als root ausgeführt werden. Wenn sudo nach einem Passwort verlangt, ist es das Passwort vom eigenen Benutzer!

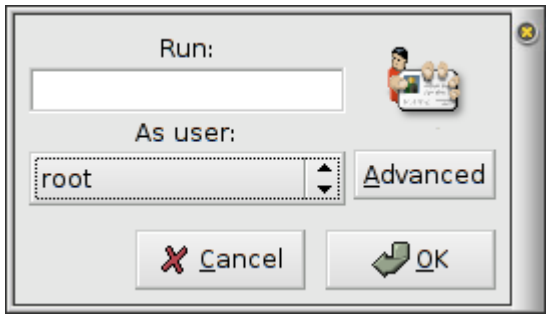
sudo → **s**witch **u**ser and **do**

Beispiel:

```
$ /usr/bin/sudo id
Password:
uid=0(root) gid=0(root) groups=0(root)
heuer@guybrush:$
```

Benutzer temporär wechseln

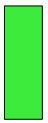
Mit dem Befehl **gksu** bzw. **kdesu** vor dem Befehl können sie auch unter den graphischen Oberflächen Gnome bzw KDE Befehle als andere Benutzer ausgeführt werden.



ACHTUNG !!!

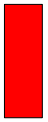
Gewöhnen sie sich an sicherheitsrelevante Programme immer mit dem absoluten Pfad aufzurufen:

```
heuer@guybrush:$ /usr/bin/sudo id
```



und nicht:

```
heuer@guybrush:$ sudo id
```



Es ist viel schwieriger in `/usr/bin/` Programme auszutauschen als beispielsweise im Homeverzeichnis von einem Benutzer!

Fragen ?

Übungen

2.4: Verwenden Sie das Programm su, um Systemverwalterprivilegien zu erlangen, und wechseln Sie zurück zu Ihrer normalen Benutzerkennung.

2.5: (Für Programmierer.) Schreiben Sie ein überzeugendes »trojanisches« su-Programm. Versuchen Sie, Ihren Systemverwalter damit aufs Glatteis zu führen.

2.6: Versuchen Sie, mit dem »Befehl ausführen . . . «-Dialog von KDE das Programm id in einer Terminalsitzung auszuführen. Kreuzen Sie dazu das entsprechende Feld in den erweiterten Einstellungen an.

Kommandointerpreter – Shell

Quizfrage: Wer führt die Befehl aus, die sie in einem Terminal eintippen haben?

Antwort: Der Computer gilt nicht, da UNIX bzw. Linux ein Multitasking System ist. Die Eingaben werden von einer Shell interpretiert und wenn sie syntaktisch korrekt sind von der Shell ausgeführt.

shell → Mantel, Aussenhaut, Muschel

Die Shell ist **DAS** Bindeglied zwischen dem Computer und dem Anwender.

Kommandointerpreter – Shell

Neben den text-orientierten Shells gibt es auch graphische Shells.

Bekannt sind beispielsweise Gnome oder KDE. Es gibt auch viele weitere Window-Manaer (Sawfisch, Enlightment, xfce, compiz, ...) Die graphischen Shells lesen neben den Tastatur-Eingaben auch die Maus-Bewegungen und -Clicks.

Shell

Bei der Entwicklung von UNIX wurden verschiedene Shells entwickelt

Bourne Shell	(sh)
C-Shell	(csh)
TENEX C Shell	(tcsh)
Korn	(ksh)
Bourne-Again-Shell	(bash)

Linux verwendet die **bash** als Standard-Shell

Mit der Eingabe von **echo $0** in einer Shell kann angezeigt werden, welche Shell man gerade verwendet.

```
$ echo $0
bash
```

Shell Skripte

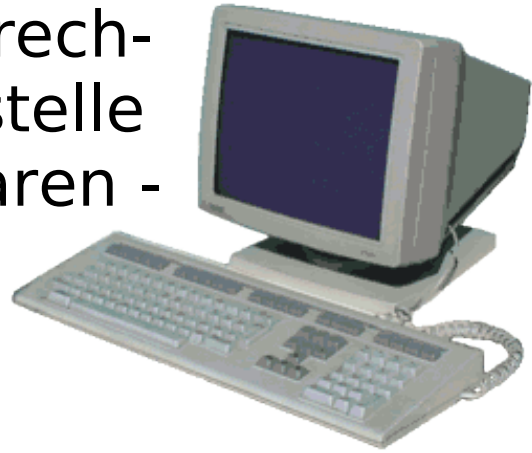
Shells können die Befehle nicht nur von der Tastatur lesen sondern auch aus Files – Shell-Skripte genannt – verarbeiten.

Der Shell-Skript muss auf der ersten Zeile folgenden Eintrag haben und das execute Flag von der Datei sollte gesetzt sein (Siehe Kapitel 7)

```
#!/bin/bash
# ein Shell-Skript
echo “hello world“
```

Terminals

Am Anfang wurde hauptsächlich auf Grossrechnern - an denen via einer seriellen Schnittstelle (RS232) ASCII-Terminals angeschlossen waren - gearbeitet.



Digital VT320 terminal

Heute wird dazu der PC als Terminal und Netzwerke für den Transport der Pakete verwendet.

An einen Linux Rechner können – serielle Schnittstellen vorausgesetzt – Terminals angeschlossen werden.

Meistens werden virtuelle Terminals verwendet. Linux bietet auf der Ebene der Console mehrere Terminals an. Sie können jeweils mit **`[ctrl]-[alt]-[Fn]`** von einem Terminal zum Anderen wechseln.

Terminals

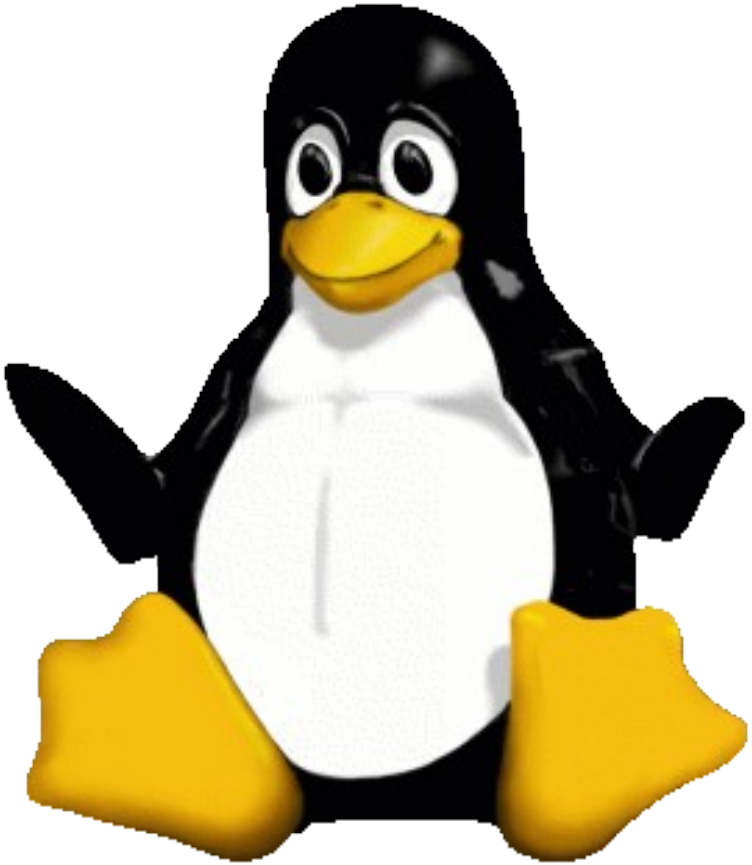
Die virtuellen Terminal 1 – 6 sind meistens normale Terminals, das virtuelle Terminal 7 ist meistens der graphische Oberfläche zugeordnet – wenn sie gestartet wurde. Einzelne Distributionen können davon abweichen!

Mit der Tastenkombination **[shift]-[PgUp]**, **[shift]-[PgDn]** kann der Inhalt des aktuellen Terminal geblättert werden.

Unter der graphischen Oberflächen werden Pseudo-Terminals und Terminal-Emulationen verwendet.

Beispiele solcher Terminal-Emulationen sind **xterm, konsole, gnome-terminal, Eterm, mrxvt, Aterm, ...**

Fragen ?



Übungen

2.7: Wie viele verschiedene Shells sind auf Ihrem Rechner installiert und welche? (Tipp: Prüfen Sie die Datei /etc/shells.)

2.8: Melden Sie sich ab und wieder an und prüfen Sie die Ausgabe des Kommandos **`echo $0`** in der Loginshell. Starten Sie mit dem Kommando **`bash`** eine neue Shell und geben Sie wiederum das Kommando **`echo $0`**. Vergleichen Sie die Ausgabe der beiden Kommandos. Fällt Ihnen etwas auf?

Kommandos

Jeder Rechner arbeitet nach den folgenden drei Schritten:

- Der Rechner wartet auf eine Eingabe durch den Benutzer.

  Unter Linux wartet eine Shell auf die Eingabe des Benutzers

- Der Benutzer legt ein Kommando fest und gibt dieses per Tastatur oder per Maus ein.

  Unter Linux nimmt die Shell die Eingabe des Benutzers entgegen.

- Der Rechner führt die erhaltene Anweisung aus.

  Unter Linux führt die Shell die Anweisung des Benutzers aus.

Aufbau der Kommandos

Damit die Shell die Eingabe des Benutzers versteht, muss der Benutzer einen Syntax, den die shell 'versteht', verwenden.

Syntax: **<kommando> <option> <argument>**

Kommando:	"Was wird gemacht"
Option:	"Wie wird es gemacht"
Argument:	"Womit wird es gemacht"

Aufbau der Kommandos

Kommando:

- ist immer das erste Wort
- ist case-sensitive, d.h. **`id`** , **`iD`** und **`ID`** sind unterschiedliche Kommandos

Es gibt Shell interne und externe Kommandos. Die Shell durchsucht die internen Kommandos vor den externen.
▶ Wenn es ein Kommando sowohl als internen wie auch als externen Befehl gibt, wird der interne Befehl ausgeführt.

Aufbau der Kommandos

Option:

- beginnen mit einem Minus '-', gefolgt von einem oder mehreren Buchstaben oder Zahlen.

  ➔ kryptische Befehle, mehrere Optionen können zusammengefasst werden
  z.B. `ls -a -b` kann als `ls -ab` geschrieben werden

- beginnen mit zwei Minus '--', gefolgt von ausgeschriebenen Wörtern

  ➔ klarere Befehle, dafür mehr Tipparbeit, Mehrere Optionen können **nicht** zusammengefasst werden.
  z.B. `ls --all --escape`

Aufbau der Kommandos

Argument:

- Ohne einleitende Zeichen
- Bedeutung je nach Kommando, in der Regel sind dabei Dateinamen gemeint

interne / externe Kommandos

interne Kommandos:

- Diese Befehle werden von der Shell selber implementiert und können sofort verwendet werden können.
  Bei der **bash** sind das ca. 30 Befehle.
  Einige Befehle sind nur als interne Befehle nützlich (**cd**, **exit**).

externe Kommandos:

- Dies sind ausführbare Dateien die im Dateisystem abgelegt sind. Das ausführen eines externen Kommandos dauert länger, weil das Betriebssystem zuerst einen neuen Prozess erzeugen muss, der dann Datei ausführen wird.
  Shell-Skripte sind externe Kommandos.

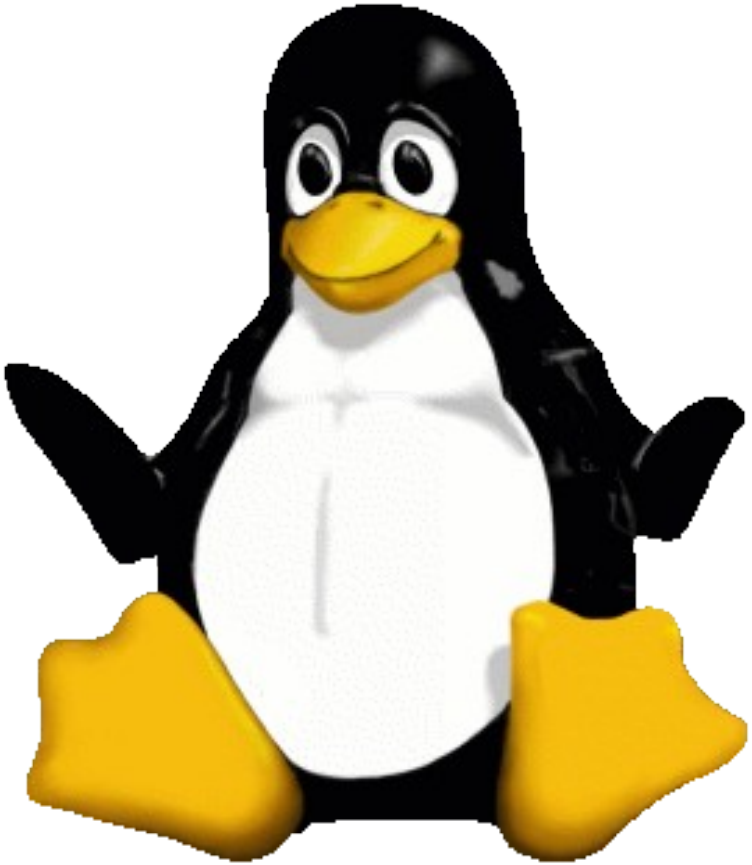
interne / externe Kommandos

Um heraus zu finden ob ein Befehl ein interner- oder externer Befehl ist, kann der Befehl **type** benutzt werden:

```
$ type cd
cd is a shell builtin

$ type echo
echo is a shell builtin

$ type /bin/echo
/bin/echo is /bin/echo

$ type passwd
passwd is /usr/bin/passwd
```

▶ Befehle, die es sowohl intern als auch extern gibt, müssen sich nicht zwingend gleich verhalten.

Fragen ?



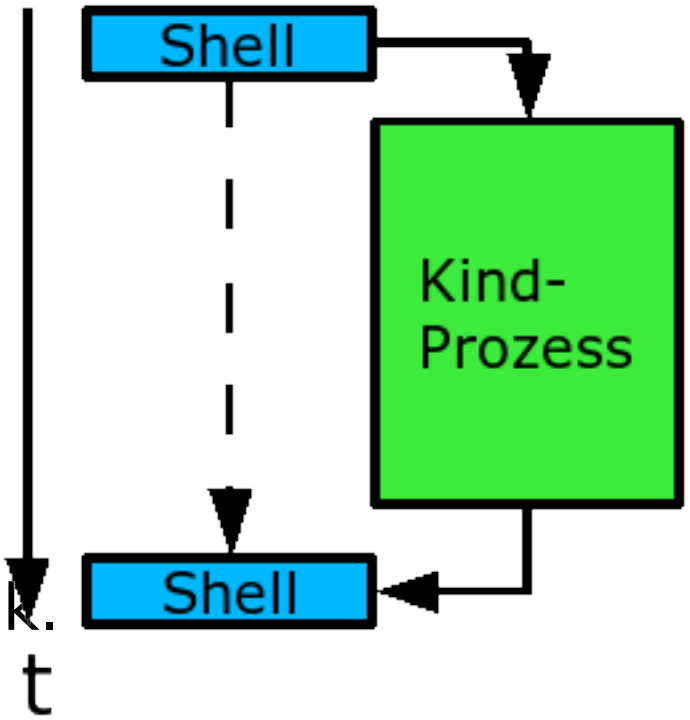
Vorder- / Hinter-Grund Prozesse

Linux ist ein Multitasking Betriebssystem und kann mehre Prozesse gleichzeitig verwalten. Die Shell kann dem Benutzer diese Fähigkeiten weiter geben.

Befehle können Synchron (im Vordergrund) oder Asynchron (im Hintergrund) ausgeführt werden.

Vordergrund Prozess

Synchron:
Der Benutzer gibt einen Befehl ein. Die Shell startet einen neuen Prozess und übergibt diesem Prozess die Kontrolle. Der Kind-Prozess führt den Befehl vom Benutzer aus.



Wenn der Kind-Prozess terminiert bekommt die Shell die Kontrolle zurück.

Während dem der Kind-Prozess ausgeführt wird, werden die Eingaben an den Kind-Prozess weiter gegeben, Die Ausgaben des Kind-Prozesses erscheinen im Terminal der Shell.

Vordergrund Prozess

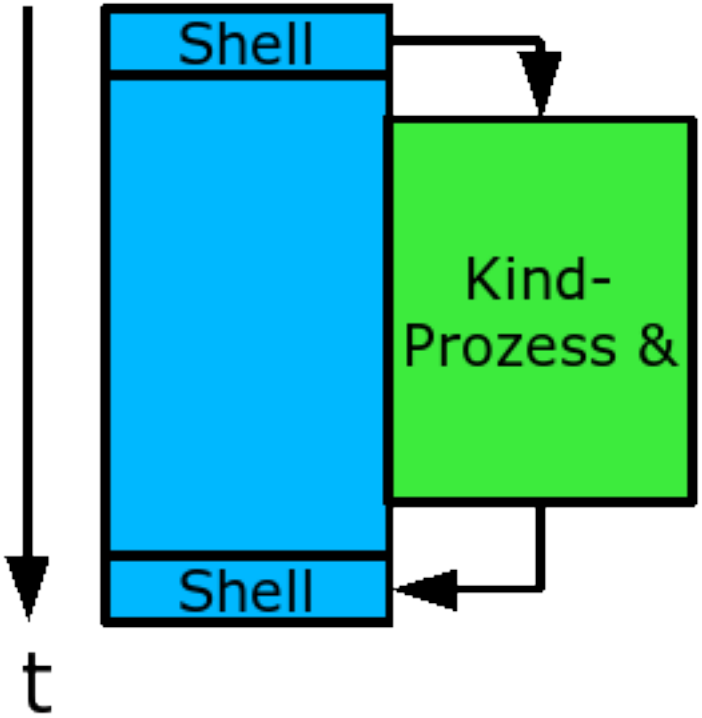
Einzelne Befehle können mit einem Semikolon (;) getrennt werden

Beispiel:

```
$ date; sleep 10; date
Sat Mar 22 16:02:22 CET 2008
Sat Mar 22 16:02:32 CET 2008
$
```

Hintergrund Prozess

Asynchron:
Der Benutzer gibt einen Befehl ein.
Die Shell startet einen neuen Prozess
behält die Kontrolle selber.
Gleich nach dem Start vom Kind-Prozess kann die Shell den nächsten Befehl entgegennehmen.



Da die Shell die Kontrolle behält, können an den Kind-Prozess keine Tastatureingaben gesendet werden.
Die Ausgaben des Kind-Prozesses erscheinen im Terminal der Shell.

Hintergrund Prozess

Beispiel:

Der Befehle der einem **&** vorangeht wird im Hintergrund ausgeführt

```
$ date; sleep 10& date
Sat Mar 22 16:05:23 CET 2008
[1] 5216
Sat Mar 22 16:05:23 CET 2008
$ jobs
[1]+  Running                 sleep 10 &
$ <return>
[1]+  Done                    sleep 10
$ <return>
```

Der Prompt ist sofort wieder da und die Shell kann den nächsten Befehl entgegen nehmen

Job Kontrolle

Vordergrundprozesse:

[ctrl]-c	Abbrechen vom aktiven Programm, das Programm kann noch selber 'aufräumen',
[ctrl]-	Abbrechen vom aktiven Programm, das Programm wird rücksichtslos 'ermordet', d.h. das Programm kann nicht mehr aufräumen
[ctrl]-z	Anhalten vom aktiven Programm
bg	Weiterlaufen des angehaltenen Programms im Hintergrund (**bg** → background)
fg	Weiterlaufen des angehaltenen Programms im Vordergrund (**fg** → foreground)

Job Kontrolle

Hintergrundprozesse:

Es sind keine Tastaturkürzel möglich, da diese den laufenden Hintergrundprozess gar nicht erreichen können!

`bg %jobid`	weiterlaufen des angehaltenen Programms im Hintergrund (**bg** → background)
`fg %jobid`	weiterlaufen des angehaltenen Programms im Vordergrund (**fg** → foreground)
`kill %jobid`	entspricht [ctrl]-c (kill -TERM %jobid)
`kill -KILL %jobid`	entspricht [ctrl]-\
`kill -STOP %jobid`	entspricht [ctrl]-z
`kill -CONT %jobid`	entspricht bg %jobid, fg %jobid,

Job Kontrolle

Der Befehl `jobs`

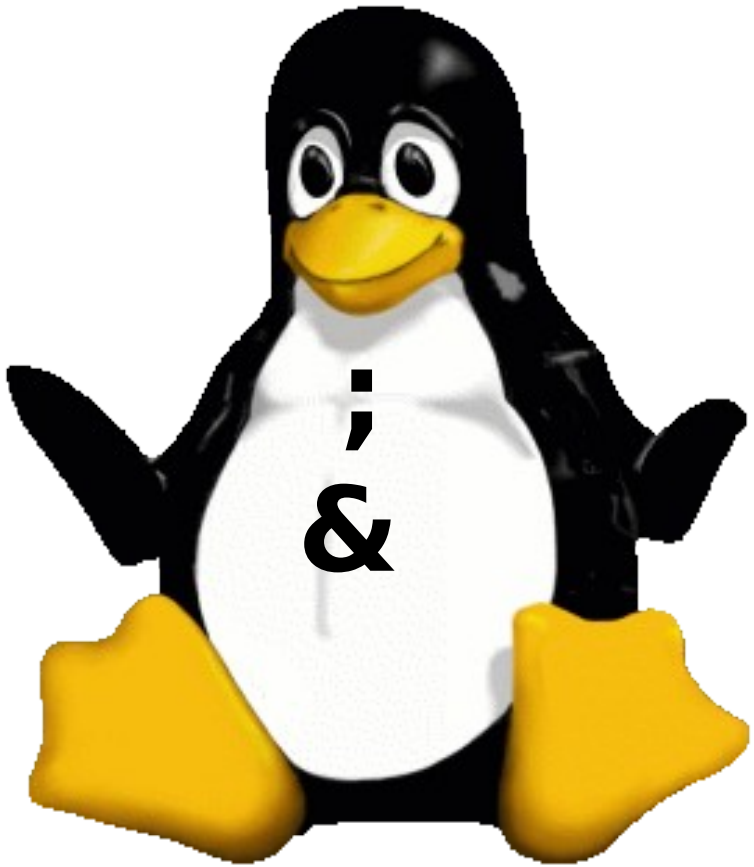
Der Befehl **`jobs`** listet die in **derselben** Shell gestarteten Hintergrund**jobs** auf.

`jobs` kennt folgende Optionen:

`-l (long)`	Zeigt zusätzlich die ProzessID (PID) an.
`-n (notify)`	Zeigt nur die Prozesse, die seit dem letzten Aufruf von **`jobs`** beendet worden sind.
`-p (process)`	Zeigt nur die PID an

`jobs` kennt keine Argumente.

Fragen ?



Übungen

2.9: Welche der folgenden Kommandos sind bei der Bash extern und welche intern realisiert:

	intern	extern	beides
`alias`			
`echo`			
`rm`			
`test`			

2.10: Verwenden Sie ein geeignet spektakuläres Programm (etwa die OpenGL-Demonstration `xgears`, alternativ etwa `xclock -update 1`), um mit Hintergrundprozessen und Jobkontrolle zu experimentieren.
Vergewissern Sie sich, dass Sie Prozesse im Hintergrund starten können, dass Sie Vordergrundprozesse mit [Ctrl]+[z] anhalten und mit bg in den Hintergrund schicken können, dass jobs die Hintergrundprozesse auflistet und so weiter.

2.11: Beschreiben (und erklären) Sie die Unterschiede zwischen den folgenden drei Kommandozeilen:

```
$ sleep 5 ; sleep 5
$ sleep 5 ; sleep 5 &
$ sleep 5 & sleep 5 &
```

Die Shell als Werkzeug

Die Shells haben sehr viele hilfreiche Werkzeuge eingebaut, die das Arbeiten mit der Shell vereinfachen.

Kommandoeditor:
Die Befehlszeile kann mit einem einfachen Editor modifiziert werden. Darum ist es nicht notwendig die ganze Zeile neu einzutippen, wenn beispielsweise Schreibfehler korrigiert werden müssen.

Kommandoabbruch:
Die Eingabe kann mit **`[ctrl]-[c]`** abgebrochen werden. Gestartete Vordergrund-Prozesse können mit **`[ctrl]-[c]`** abgebrochen werden oder mit **`[ctrl]-[\]`** 'abgeschossen' werden.

Die Shell als Werkzeug

Historyfunktion:
Mit **[ctrl]-[r]**kann in der Befehlshistory nach alten Befehlen gesucht werden.

Autovervollständigung:
Mit **[tab]**oder **[ctrl]-[i]** kann ein Befehl/Dateiname vervollständigt werden. Ist der vorgegebene Dateiname nicht eindeutig, so wird beim ersten **[tab]** soweit ergänzt wie möglich, wird dann **[tab]** nochmals gedrückt, werden alle passenden Möglichkeiten aufgelistet.

Scrollen:
Mit **[shift]-[PgUp]** oder **[shift]-[PgDn]** kann auf der Console oder Terminal-Emulationen seitenweise gescrollt werden.

Tastaturkürzel

Tastaturkürzel: (Zusammenfassung)

[↑], [↓]	Durch die Befehls-History scrollen
[←], [→]	in der Befehlszeile hin und her scrollen
[ctrl]-[r]	In der Befehls-History suchen
[ctrl]-[a], [Home]	Springe an den Anfang der Befehlszeile
[ctrl]-[e], [End]	Springe an das Ende der Befehlszeile
[ctrl]-[t]	Die Zeichen vor und unter dem Cursor vertauschen
[ctrl]-[h], [Backspace]	Zeichen links vom Cursor löschen
[Delete]	Zeichen unter dem Cursor löschen
[ctrl]-[l]	Bildschirm löschen
[ctrl]-[i], [Tab]	Vervollständigen des Befehls oder Dateinamens
[ctrl]-[c]	Befehlseingabe abbrechen, Befehl abbrechen
[ctrl]-[\]	Laufenden Befehl abwürgen
[shift]-[PgUp], [shift]-[PgDn]	Scrollen innerhalb der Console oder Terminal-Emulation

Die Shell als Werkzeug

Kommandos umbenennen:
Befehle können mit **`alias`** umbenennt werden. Damit kann man sich die Arbeit erleichtern, aber man kann auch das ganze System "auf den Kopf" stellen. Passen sie auf, wenn sie einen **`alias`** definieren!

Mit dem Befehl **`unalias <alias>`** kann ein Alias in der aktuellen Shell permanent gelöscht werden.

Will man verhindern, dass der Alias nur beim nächsten Befehl nicht ausgeführt wird, so kann man dem Befehl ein Backslash (\) voranstellen.

Die Shell als Werkzeug

Beispiele für den Befehl `alias`

Ein beliebter **`alias`** ist beispielsweise
`alias rm='rm -i'`
Damit wird der Befehl **`rm`** (remove) immer mit der Option **`-i`** (interaktiv) ausgeführt

`unalias rm`
Nun wird der Befehl **`rm`** wieder normal ausgeführt

`\rm`
Der Befehl **`rm`** wird direkt - ohne dass die Alias-Definition interpretiert werden - ausgeführt.

Variablen

Variablen:
In der Shell können sie Variablen mit
```
$ variable="Das ist ein Wert.";
```
erzeugen und einen Wert zuweisen.

Um die Variable auszulesen muss ein Dollarzeichen vorangestellt werden:
```
$ echo $variable
Das ist ein Wert.
```

Der Befehl **env** zeigt alle definierten Variablen an.

Variablen

Die Shell kennt zwei Variablen-Bereiche

Local:
Die Variablen hier sind nur innerhalb der Shell gültig.

Environment:
Die Variablen werden an einen neu gestarteten Prozess vererbt. Das Programm, das von dieser Shell (genauer Prozess) gestartet wird, kann die Variablen in seinem eigenen getrennten Environment finden.

Environments Variablen werden normalerweise in Grossbuchstaben geschrieben. (Dies ist nicht zwingend, es hat sich so eingebürgert)

Variablen der Shell

Die Shell erzeugt folgende Environments Variablen:

SHELL	Pfad zur aktuellen Shell
TERM	Bezeichnung des Terminal um die richtigen Steuersequenzen verwenden zu können.
USER	Der aktuelle User
MAIL	Pfad zur Mailbox (obsolete)
PATH	Such-Pfad um Programme zu finden
PWD	Aktuelles Verzeichnis (**P**rint **W**orking **D**irectory)
SHLVL	Shell-Level (Wie viele Shells sind ineinander gestartet)
HOME	das eigene Homeverzeichnis (entspricht auch ~)
LOGNAME	Name des Users, der die shell startete, Entspricht in der Regel der Variable USER
_ (underscore)	Pfad vom letzten ausgeführten externen Programm

Diese Variablen können in Skripten oder Programme verwendet werden.

Variablen

Damit eine lokale Variable ins Environment übernommen wird, muss diese mit **`export <variablenname>`** exportiert werden.

```
export variable variable2
```

Nur die Variable und nicht $variable angeben!

Variablen können mit **`unset variable`** gelöscht werden.

Die Shell als Werkzeug

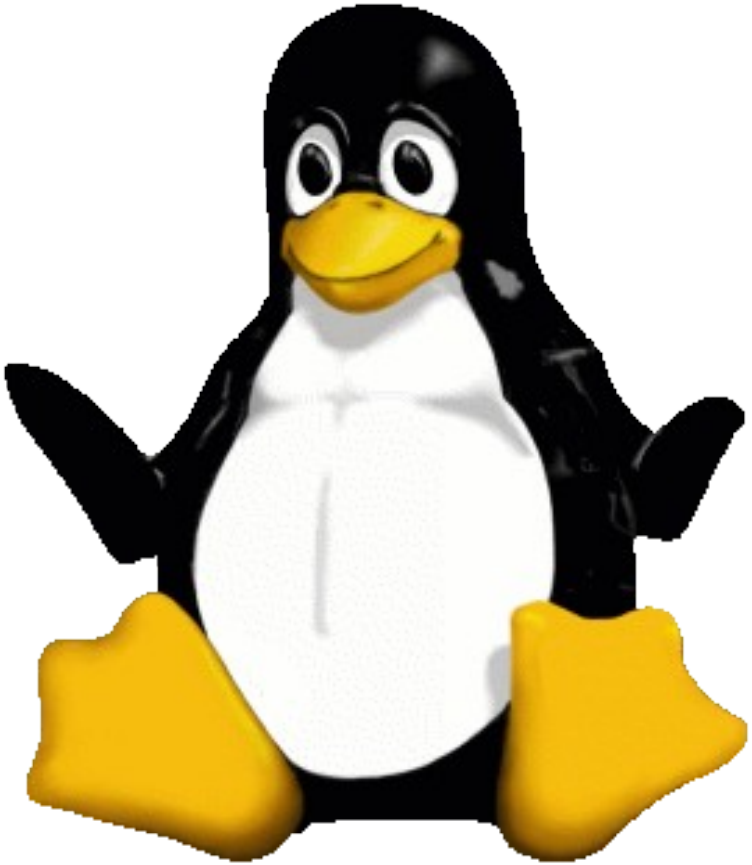
Die Aliase, Environment oder History sind nur solange aktiv wie die Shell aktiv ist. Will man diese dauerhaft verwenden, so müssen die Aliase und Environment-Variablen in einem der verschiedenen Konfiguration-Dateien der entsprechenden Shell eingetragen werden.

`~/.bash_profile`, **`~/.bash_login`**, **`~/.profile`**, **`~/.bashrc`** Benutzer spezifische Files, In der Regel wird nur eines dieser Files eingelesen. Wenn sie sicher gehen wollen, dass alle möglichen Files ausgeführt werden, müssen sie diese in `.bash_profile` includen. Diese Files kann jeder Benutzer selber verwalten und sind nur für den jeweiligen Benutzer gültig.

`/etc/profile`, **`/etc/bash.bashrc`** Systemweite Konfigurations Datei. Die Eintragungen hier werden von allen Benutzern verwendet.

`~/.bash_logout` Bei einer interaktiven Shell wird diese Datei ausgeführt, wenn sich der Benutzer abmeldet. Das ist der richtige Ort, um beispielsweise den Bildschirm mit **`clear`** zu löschen.

Fragen ?



Sonderzeichen

Bei den Variablen haben wir gesehen, dass wenn man einem Namen eine Dollar-Zeichen voranstellt der Name zu einer Variablen wird. Das Dollar-Zeichen wird von der Shell als Sonderzeichen interpretiert.

Die Shell interpretiert die folgenden Zeichen speziell:
[space]\'“$&;(){}[]*?!<>-

Wenn sie Dateinamen bzw. Argument verwenden wollen, die diese Zeichen enthalten, müssen sie verhindern, dass diese Zeichen von der Shell interpretiert werden.

Sonderzeichen

Einfache Anführungszeichen (') verhindern jegliche Interpretation. d.h. Variablen werden nicht erkannt.

Doppelte Anführungszeichen (") erkennt die Variablen und expandiert diese. Leerzeichen **`[space]`** werden durch die doppelten Anführungszeichen nicht als Trennzeichen erkannt.

Ein Backslash (\) verhindert, dass das nachfolgende Zeichen als Sonderzeichen erkannt wird.

Sonderzeichen Beispiele

```
$ datum="`date +%Y%m%d`"
$ echo $datum
20080416
$ touch 'Neue Datei $datum'
$ touch "Neue Datei \$datum"
```

Beide Befehle erstellen eine Datei mit dem Namen "Neue Datei $datum"

```
$ touch "Neue Datei $datum"
$ touch Neue\ Datei\ $datum
```

Beide Befehe erstellen eine Datei mit dem Namen "Neue Datei 20080416"

```
$ touch Neue Datei $datum
```

erstellt 3 Dateien: "Neue", "Datei", "20080416"

Suchmuster (Shell)

Die meisten Shells erlauben es Gruppen von Datei durch ein Suchmuster zu beschreiben.
Es gelten folgende Regeln:

- Ein Fragezeichen '**?**' steht für ein beliebiges Zeichen
- Ein Stern '*' steht für beliebig viele, beliebige Zeichen
- Zeichen in eckigen Klammern **[abc]** stehen für genau ein Zeichen aus dieser Klasse (hier "a", "b" oder "c"). Bereiche in der Form **[A-Za-z]** sind erlaubt.

  Ist das erste Zeichen in der Klasse ein Ausrufungszeichen, steht die Klasse für alle nicht aufgeführten Zeichen.

Suchmuster (Shell)

- Dateinamen die mit einem Punkt (.) beginnen sind versteckte Dateien. Diese werden nur gefunden, wenn das Suchmuster auch mit einem Punkt beginnt.

Die Angaben in den eckigen Klammern werden abhängig von den locals (Spracheinstellungen) interpretiert. Es kann daher sein, dass dabei die grossen und kleinen Buchstaben gleichgesetzt sind.

Beispiele:

`ls a*`	Findet alle Dateien, die mit einem a beginnen
`ls ?[abc]*`	findet alle Dateien, die im Namen an 2. Stelle ein a,b oder c enthalten
`ls *[ch]`	Findet alle Dateien, deren Namen mit ein c oder h enden
`ls .m*`	Findet alle versteckten Dateien, die mit m beginnen.
`ls [A-H]*`	Findet alle Dateien, die nicht mit einem grossen Buchstaben beginnen (Hier schlagen die Locals-Settings zu)

`ls *[0-9]*.ods`

Findet alle Dateien, die im Namen eine Ziffer enthalten und auf .ods enden

Suchmuster (Shell)

Findet die Shell passende Dateien, so ersetzt **die Shell** das Suchmuster mit allen passenden Dateinamen und startet dann den Befehl – mit den Dateinamen als Argument.

Findet die Shell keine passende Dateien, so wird der Befehl mit dem Suchmuster als Argument gestartet.

- Einfache Anführungszeichen (') verhindern, dass das Suchmuster verwendet wird.
- Doppelte Anführungszeichen (") sind notwendig, wenn Leerzeichen in dem Suchmuster enthalten sind.
- Ein Backslash (\) verhindert, dass die Shell das nachfolgende Zeichen interpretiert.

Suchmuster (Shell) Beispiele

```
$ ls f*
frosch.txt
$ ls *h*
frosch.txt      teilnehmer0.dat
teilnehmer.dat  teilnehmer1.dat
$ ls [fz]
ls: [fz]: No such file or directory
$ ls [fz]*
frosch.txt  zeiten.dat
$ ls [!fz]*
regex.txt       teilnehmer0.dat
teilnehmer.dat  teilnehmer1.dat
$ ls *txt
frosch.txt  regex.txt
$ ls *x.*
regex.txt
$
```

```
frosch.txt
regex.txt
teilnehmer.dat
teilnehmer0.dat
teilnehmer1.dat
zeiten.dat
```

Fragen ?

Übungen

2.12: Definieren Sie ein Alias hw, das die Nachricht "Hallo Welt" auf dem Bildschirm ausgibt.

2.13: Sorgen Sie dafür, dass ihnen dieses Alias auch nach dem Ab- und Wiederanmelden noch zur Verfügung steht.

2.14: Was ist das Problem mit dem Kommando »echo "Hallo!"«?
(Tipp: Experimentieren Sie mit Kommandos der Form "!-2" oder "!ls".)

2.15: Erklären Sie den Unterschied zwischen den drei folgenden Bash-Kommandozeilen:

```
$ echo $HOME/*
$ echo "$HOME/*"
$ echo '$HOME/*'d
```

Übungen

2.16: Wechseln sie mit **`cd /usr/share/doc`** das Verzeichnis.
Suchen sie nun die kürzesten Suchmuster um mit dem Befehl **`ls -d XXXX`** folgende Resultate zu finden: Ersetzen sie XXXX mit der entsprechenden Suchmaske.

Beispiel:
`knoppix@Knoppix:/usr/share/doc$` **`ls -d konq*`** ergibt dann folgendes Resultat:

```
konq-plugins  konqueror  konqueror-nsplugins
```

a) debfoster diff diffstat dosfstools
b) iputils-tracepath kdebluetooth
c) ext2resize imaze