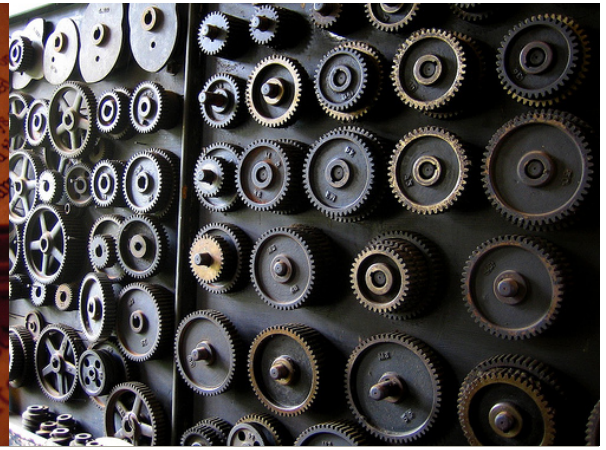
Freie Universität Berlin



Systemverwaltung Sommersemester 2009
am Fachbereich Mathematik und Informatik

Shell Scripting

Daniel Bößwetter <daniel.boesswetter@fu-berlin.de>
August 2009

Motivation / Inhalt

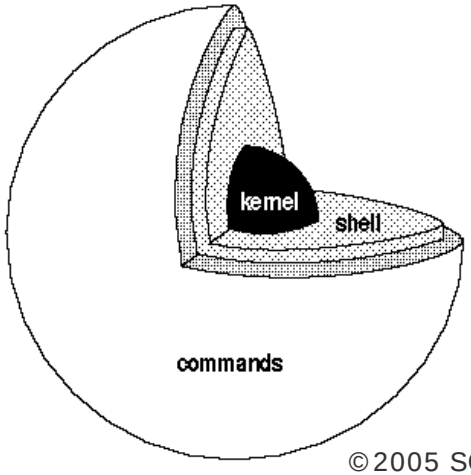
-Worum geht‘s?

-Was ist

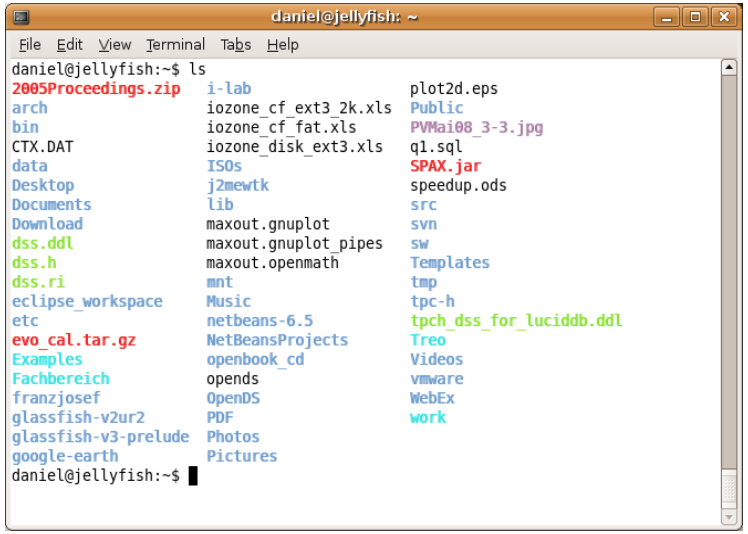
- eine Shell?
- ein Script?

-Warum machen wir nicht Java?

-Vorkenntnisse?



©2005 SCO Group



Bildnachweis (Titelfolie):
http://www.flickr.com/photos/rogersmith/
http://www.flickr.com/photos/harshadsharma/
http://www.flickr.com/photos/tin-g/

Überblick

Mi. 05.08	**Do. 06.08**	**Fr. 07.08**
Unix-Crashkurs (P)	Unix Text-Tools (V)	HTTP und CGI (V)
Einführung in Unix (V)	Logfile-Analyse (P)	CGI-top (P)
Pause	**Pause**	**Pause**
Unix Filesystem (V)	Unix Prozesse (V)	Dies und das (tar/gz, find, Windows Powershell …)
Scripting Übung (P)	Parallele Log-Analyse (P)	

Vormittags 10:00-12:30

Nachmittag 13:30-16:00

Crashkurs Unix

Aufgabe 1: Einrichten des Webservers

-Einloggen auf Eurer Kurs-Maschine (ssh)

-Erzeugen der Verzeichnis-Struktur

- ~/apache2
  - htdocs
  - var
    - log
    - lock
    - run
- rekursive Kopie von /etc/apache2/* in ~/apache2/etc (Symlinks erhalten!)
- etc/mods-enabled/cgid* löschen

-Umgang mit vi lernen (vimtutor)

-Konfiguration editieren

- Von jeder editierten Datei eine Sicherungs-Kopie anlegen (datei~ oder datei.bak)
- Absolute Pfade auf die Kopie anpassen (apache2.conf und sites-available/*)
- Portnummer >1024 in ports.conf

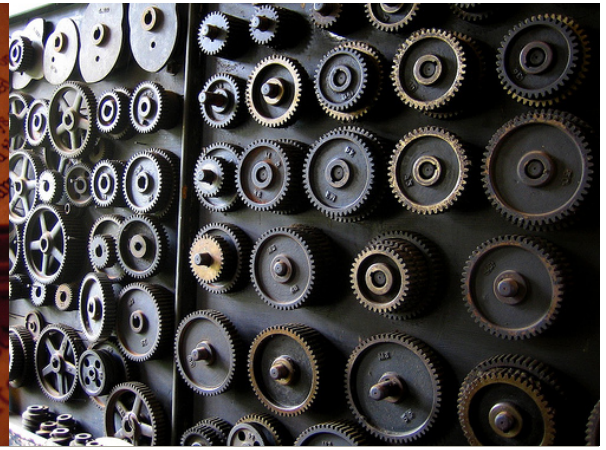
-Server-Start im ServerRoot mit /usr/sbin/apache2ctl –f etc/apache2.conf

-Apache-Doku ins htodcs entpacken

Wichtige Kommandos

man	Manualseite lesen (Beenden mit „q“)
pwd cd <dir>	Aktuelles Verzeichnis ausgeben/wechseln
ls	Verzeichnisinhalt auflisten
head, tail (-f)	Dateianfang/-ende
more, less, cat	Text-Dateien lesen, konkatenieren
mkdir, rmdir	Leeres Verzeichnis anlegen/löschen
cp, mv, rm	Datei kopieren/verschieben (umbenennen)/löschen
ln	(symbolische) Links anlegen
vi / vim	Text-Editor
tar	Archivierungs-Programm

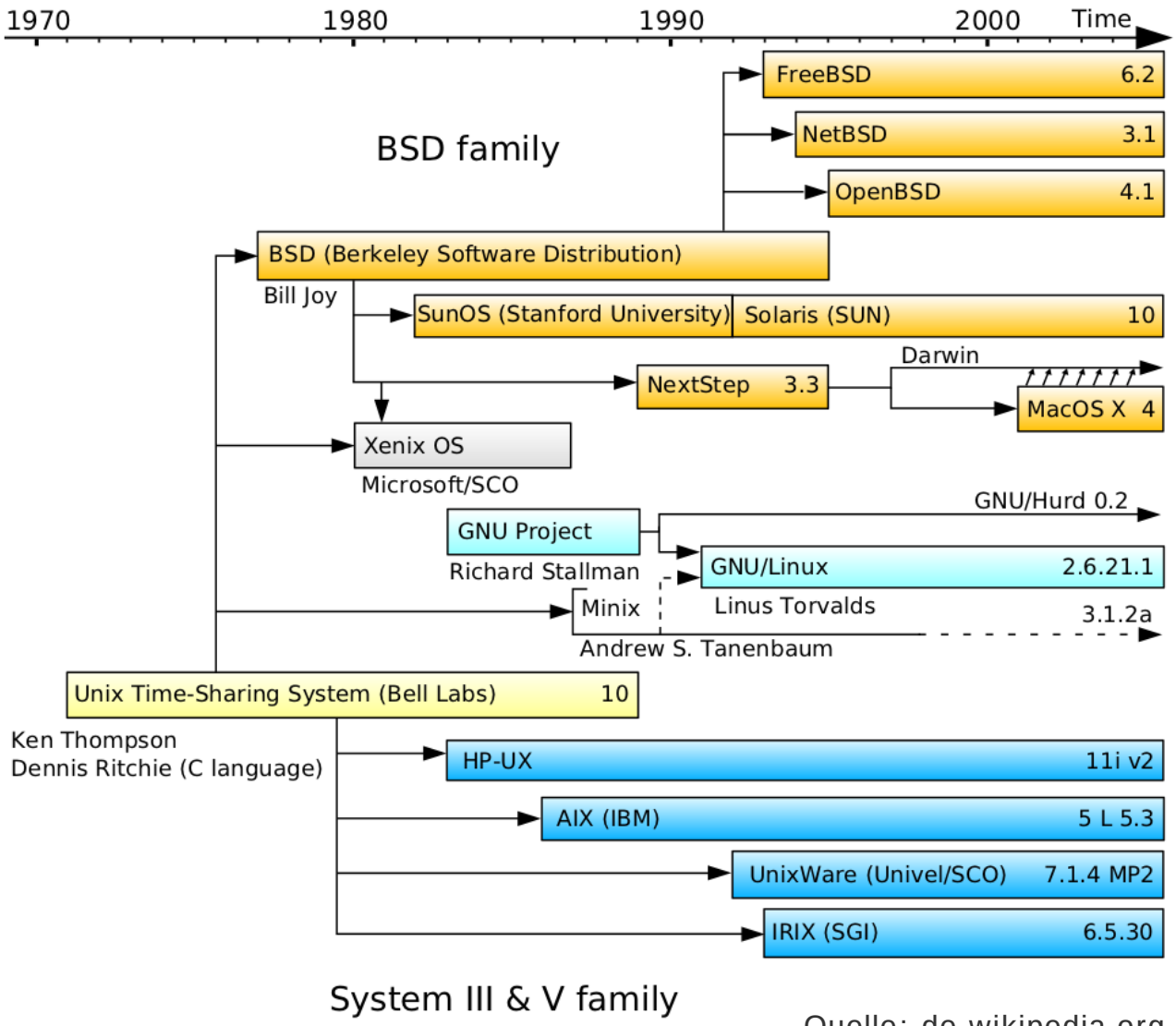
Freie Universität Berlin



Unix

„*Unix sagt niemals ›bitte‹.*“ – Rob Pike

Unix Historie



Quelle: de.wikipedia.org

Unix Philosophie

„Mach nur eine Sache, aber mach sie gut.“

-hochspezialisierte Kommandos (Filter)

-Kombinierbarkeit der Kommandos (Pipes)

Datei- und Austauschformat ist Text

„Alles ist eine Datei“

Offene Standards

Portierbarkeit, HW-Unabhängigkeit

Multiuser, Multitasking

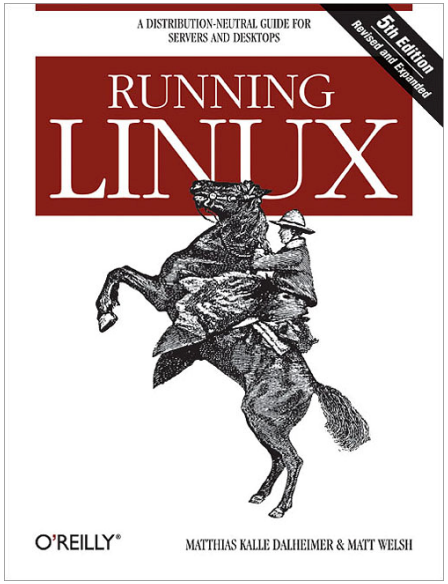
Ken Thompson, Dennis Ritchie

Linus Torvalds, Andrew S. Tannenbaum

Unix Literatur

Suche nach „Unix“ auf amazon.de liefert 898 Ergebnisse …
Eine gute Einführung in Linux ist z.B.

Running Linux, Fifth Edition
BY MATTHIAS KALLE DALHEIMER, MATT WELSH
FIFTH EDITION DECEMBER 2005
O‘REILLY



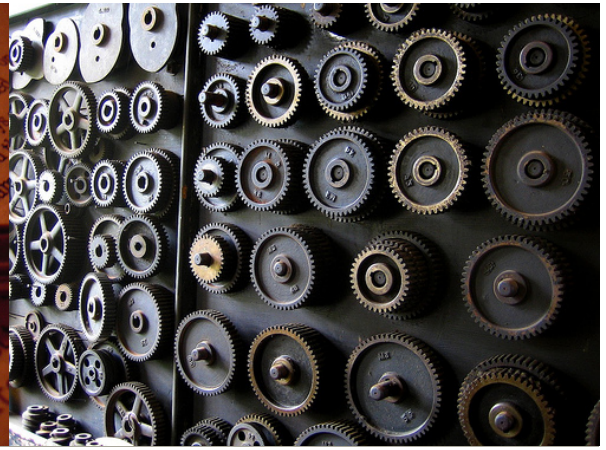
(Gibt‘s bei books.google.de!!!)

Man muss aber kein Geld ausgeben, denn:
-es gibt Skript und Folien unter
http://page.mi.fu-berlin.de/boesswet/

-Das *Linux Documentation Project* hat
mehrere Bücher online unter http://www.tldp.org

-Das Linux Anwenderhandbuch und Leitfaden für die Systemverwaltung
(Sebastian Hetze, Dirk Hohndel, Olaf Kirch, Martin Müller)
http://www.linux-ag.de/linux/LHB/LHB.html

Freie Universität Berlin



Arbeiten mit Unix-Shells

Einführung, interaktives Arbeiten, Manualseiten, Variable, Expansion, einfache und komplexe Kommandos

Unix-Shells

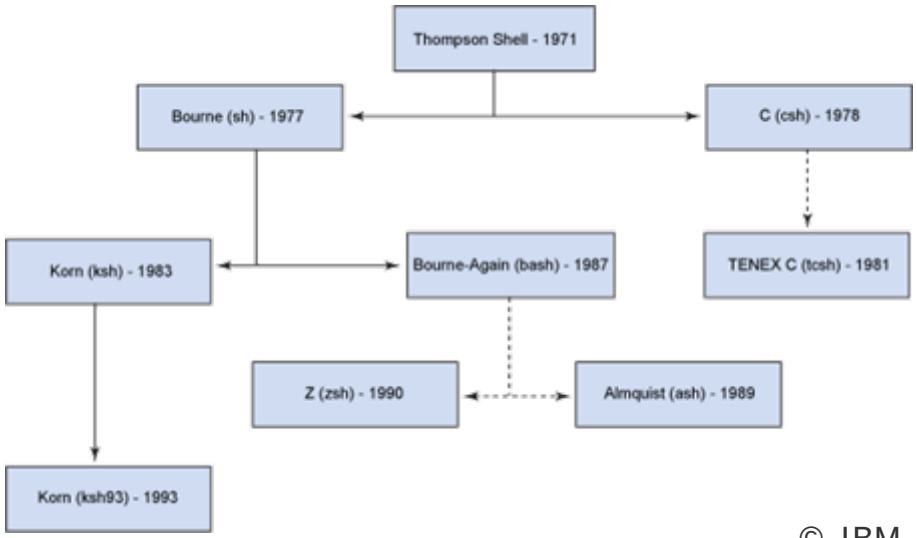
Kommandozeilen-Interpreter, primäre Benutzerschnittstelle **bevor** es grafische Oberflächen gab.

Sowohl interaktiv bedienbar als auch als Skriptsprache verwendbar

2 große Familien von Shells:

-Bourne-Shell kompatibel
- Sh / POSIX-Shell
- **Bash (wird in dieser Lehrveranstaltung verwendet)**
- Korn-Shell
- Z-Shell

-C-Shell kompatibel
- Csh
- Tcsh



© IBM

Unix-Kommandos

Was sind Unix-Kommandos?

-Einfache Befehle
- Shell-intern
- Ausführbare Programme

-Zusammengesetzte Befehle
- Pipelines
- Sequenzen
- Schleifen, Verzweigungen, …

-Optionen und Argumente
- kurze Optionen, z.B. *–n* (ein Minus, ein Buchstabe)
- lage Optionen, z.B. *--force* (zwei Minus und mehrere Buchstaben)
- Optionen können Argumente verlangen, z.B. *–o file*
- Manche Argument werden direkt übergeben, z.B. Dateinamen

-numerischer Return-Code $?
- 0 für OK, Fehler sonst

Weitere wichtige Kommandos

find	Dateien finden
file	Dateityp-Erkennung
id, whoami	Eigene Identität
hostname	Rechnername
uname	OS-Informationen
which	Suchpfad durchsuchen
type	Typ eines Kommandos herausfinden
echo	Text ausgeben
date	Datum / Uhrzeit
grep	Textdateien durchsuchen

Bash – interaktives Arbeiten

- **Kommandozeilen-Editor**
  - Pfeiltasten auf/ab für Historie (history)
  - Pfeiltasten rechts/links, Pos1, End für Bewegung innerhalb der Zeile
  - <tab> vervollständigt Kommando bzw. Dateiargumente

- **Manualseiten, man-Kommando**
  - **Sektionen des Manuals**
  - **Drucken von Manualseiten**

- **Control-C bricht laufende Kommandos ab**
- **Control-D sendet EOF**

Bash – Variable, Expansion und Maskierung

Variable

-2 Varianten mit kleinen Differenzen

- Shell-Variable: nur in der Shell verwendbar
- Environment-Variable
  - werden an Kindprozesse vererbt Beispiel: DISPLAY
  - werden in der Bash mittels export zu Environment-Variablen gemacht

-entstehen durch Zuweisung

-sind nicht typisiert

-Expansion des Variablen-Inhalts durch

- voranstellen eines $ vor den Namen
- im Zweifelsfalle ${name}

-Verhinderung der Expansion durch Maskierung (Baskslash oder einfache Hochkomma)

Beispiele

```
daniel@jellyfish:~$ shell_var="Hallo Shell"
daniel@jellyfish:~$ echo shell_var
shell_var
daniel@jellyfish:~$ echo $shell_var
Hallo Shell

daniel@jellyfish:~$ ENV_VAR="Hallo Env"
daniel@jellyfish:~$ export ENV_VAR
daniel@jellyfish:~$ echo $ENV_VAR
Hallo Environment
daniel@jellyfish:~$ echo \$ENV_VAR
$ENV_VAR
daniel@jellyfish:~$ echo '$ENV_VAR'
$ENV_VAR
daniel@jellyfish:~$ echo "$ENV_VAR"
Hallo Environment
```

Weitere Formen von Expansionen

-Shell-Globs *, ?
- ? kann für ein beliebiges Zeichen stehen
- * kann für beliebig viele beliebige Zeichen stehen

-Tilden-Expansion ~
- ~ steht für das eigene Home
- ~user steht für users Home

-Arithmetische Ausdrücke $((…))

-Sequenzen {1..20}

-Kommando-Substitution $(…) oder Backticks `…`

gelb: geht nur in der Bash

Beispiele

```
boesswet@moskau:~$ ls /bin/l*
/bin/ln  /bin/loadkeys  /bin/login
/bin/ls  /bin/lsmod
boesswet@moskau:~$ ls /bin/l?
/bin/ln  /bin/ls

boesswet@moskau:~$ echo ~
/home/datsche/boesswet
boesswet@moskau:~$ echo ~stucki
/home/ada/stucki

boesswet@moskau:~$ echo $(( 5 * 5 ))
25

boesswet@moskau:~$ x=$(date)
boesswet@moskau:~$ echo $x
Di 4. Aug 10:54:51 CEST 2009
```

Übung: Variable und Expansion

Finde heraus, wie man in der bash(1) den Prompt verändert und setze den Prompt so, dass er immer folgendes Format hat:

2001-08-01-09:31:45: usr >

Wobei am Anfang das aktuelle Datum und die Uhrzeit stehen und „usr“ das letzte Pfadelement des aktuellen Arbeitsverzeichnisses ist (hier könnte der User sich also in /usr oder in /server/usr befinden).

Tipp: für den Pfad-Anteil könnte das Konstrukt ${parameter##word} hilfreich sein (siehe bash(1)). Das Datum kann mittels date(1) ermittelt werden.

Bash – komplexe Kommandos

Sequentielle Ausführung: Aneinander-Reihung mit „;“

Bedingte Ausführung durch Lazy-Evaluation

- logisches UND &&
- logisches ODER ||
- Es werden die Return-Codes der jeweiligen Kommandos verundet bzw. verodert und ein entsprechender Return-Code generiert

Pipelines: Ausgabe eines Kommandos als Eingabe für das nächste |

Beispiele

```
$ date ; sleep 10 ; date
Tue Aug  4 12:07:09 CEST 2009
Tue Aug  4 12:07:19 CEST 2009

$ test -f /tmp/testfile || \
          touch /tmp/testfile
$ ls -l /tmp/testfile
-rw-r--r-- 1 daniel daniel 0 2009-08-04
12:08 /tmp/testfile
$ test -f /tmp/testfile || \
          touch /tmp/testfile
$ ls -l /tmp/testfile
-rw-r--r-- 1 daniel daniel 0 2009-08-04
12:08 /tmp/testfile
$ test -f /tmp/testfile && rm /tmp/testfile
$ rm /tmp/testfile
rm: cannot remove `/tmp/testfile': No such
file or directory
$ test -f /tmp/testfile && rm /tmp/testfile

$ ssh moskau tar cf - tmp | tar tf -
tmp/
tmp/09-06-07.pdf
…
```

Bash – Kontrollstrukturen

Bedingte Ausführung

if <cmd> ; then … ; else … ; fi

case <word> in <pattern>) … ;; <pattern2>) … ;; *) … ;; esac

Schleifen

for var in <list> ; do … ; done

while <cmd> ; do … ; done

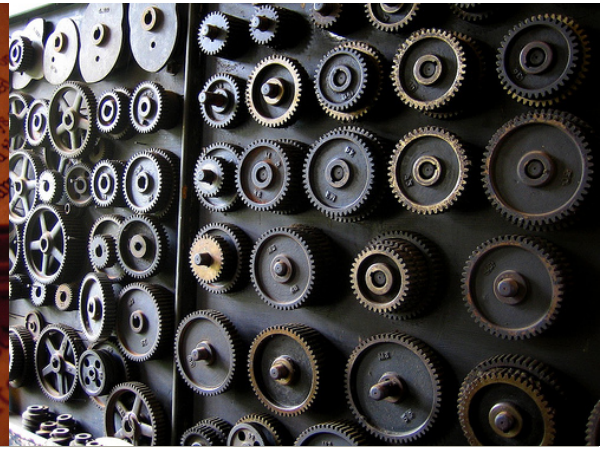
Beispiele

```
$ if [ -r /etc/shadow ] ; then \
          echo yes; else echo no ; fi
no
$ if [ -r /etc/passwd ] ; then \
          echo yes ; else echo no ; fi
yes

$ case $(date) in Tue*) echo "Tuesday" ;; \
          *) echo "Not Tuesday" ;; esac
Tuesday

$ for s in /tmp/* ; do \
  test -f $s && echo "$s is a file" ; done
/tmp/tmp.xZwTyg6298 is a file

$ while true ; do date ; sleep 3 ; done
Tue Aug  4 14:48:38 CEST 2009
Tue Aug  4 14:48:42 CEST 2009
Tue Aug  4 14:48:45 CEST 2009
```

Wichtige Kommandos

xargs	Kommandos auf Dateiliste aus Pipe ausführen
tee	Pipeline mitprotokollieren
touch	Datei anlegen/Datums-stempel anpassen
sleep	Sekundenschlaf
seq	Aufzählung
test []	Vergleiche und Datei-Tests
expr	Ausdrücke auswerten
read	Zeile lesen
true, false	tut nichts und liefert Return-Code (0 bei true, 1 bei false)

Freie Universität Berlin



Shell Scripting

Scripte editieren und ausführbar machen

Shell-Scripte

Shell-Scripte sind Textdateien, die ...

-... Kommandos enthalten, die von der Shell verstanden werden

-... als erste Zeile einen Shell-Bang mit dem Pfad zum entsprechenden Interpreter haben (z.B. #!/bin/bash)

-... Ausführbar sind (chmod +x script.sh)

Wenn das Script nicht im Suchpfad ($PATH) liegt: ausführen mit Pfadangabe!

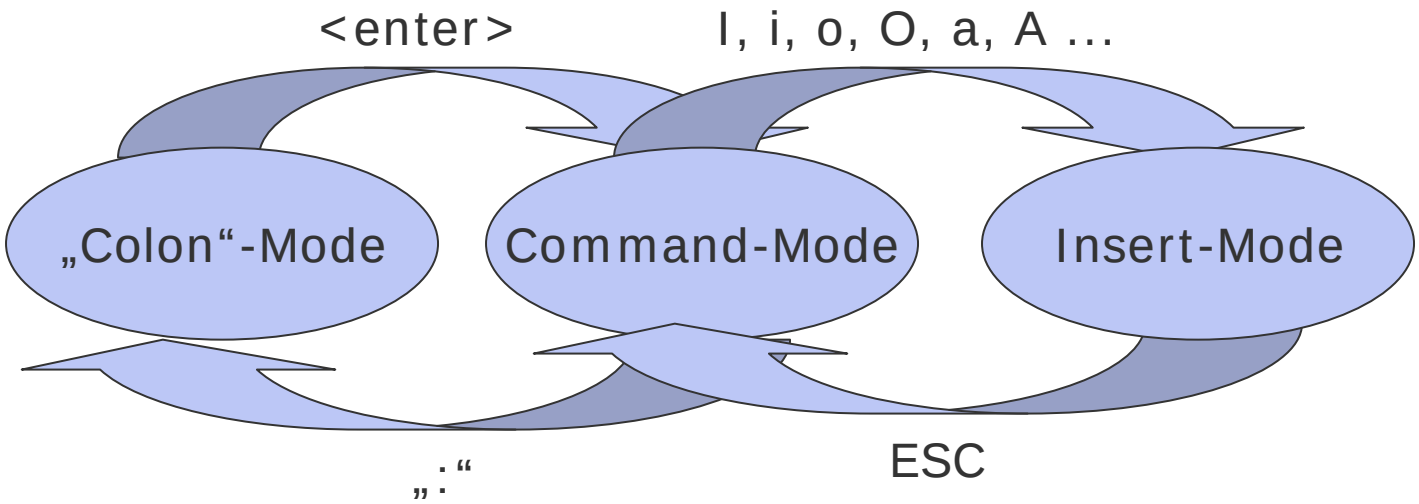
Kommandozeilenargumente $1 .. $9 bzw. $*

Texte editieren: natürlich mit vi(m)! (Der sagt übrigens auch nie ‚bitte‘.)

Kommando	**Bedeutung**
echo	Gibt Text aus
sh –x	Shell-Trace
chmod +x	Macht eine Textdatei ausführbar

```
daniel@jellyfish:~$ cat test.sh
#!/bin/bash

echo "Hallo $1!"

daniel@jellyfish:~$ ./test.sh Welt
bash: ./test.sh: Permission denied
daniel@jellyfish:~$ chmod +x test.sh
daniel@jellyfish:~$ ./test.sh Welt
Hallo Welt!
```

Vi Crashkurs



-Start des Editors mit „vi datei“ oder „vim datei“

-Nach dem Start befindet sich der Editor im Kommando-Modus

-Man begibt sich durch eines der Einfüge-Kommandos in den Einfüge-Modus.

-Mit der ESC-Taste verlässt man den Einfüge-Modus wieder

-Mit „:“ im Kommando-Modus schaltet man in den Colon- oder ex-Modus und kann in der untersten Zeile Befehle eingeben.

- Speichern mit „:w“
- Beenden mit „:q“ oder „:q!“ (wirklich ohne speichern beenden)
- Speichern und beenden mit „:wq“

-Es gibt eine Referenz-Karte unter http://tnerual.eriogerg.free.fr/vimqrc.pdf (Google nach „vim refcard“)

VIM: Kommandos

-Kommando-Modus

- **/**: Suchen
- **v**: Text markieren mit
  SHIFT-V: Zeilen markieren
- **q**: Makros aufzeichnen
- **y**: markierten Text kopieren
  yy: aktuelle Zeile kopieren
- **p**: kopierten Text einfügen

-Colon-Mode

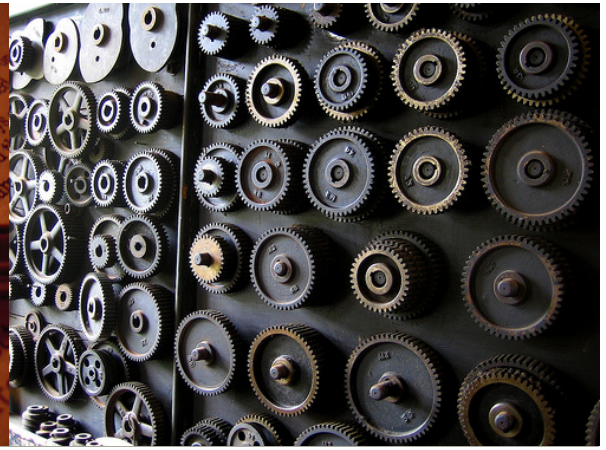
- **:%s/suchtext/ersatz/g**

  Ersetzt *suchtext* durch *ersatz* (%: alle Zeilen, g: alle Vorkommisse einer Zeile)

- **:w**
  :w! (ohne Fragen sichern)
  :wq (Sichern und beenden)
  :x (Sichern und beenden)

- **:q** (beenden)
  :q! (ohne Fragen beenden)

Freie Universität Berlin



Das Unix Dateisystem

Definition, Attribute, Berechtigungen, Kommandos

Unix Dateisystem

(Fast) alles unter Unix lässt sich als Datei betrachten

-Daten-Dateien

-Peripherie-Geräte (/dev)

-Prozesse

- Named Pipes
- /proc

-Netzwerk-Verbindungen und IPC

- Offene Sockets können immer wie offene Dateien verwendet werden.
- Named Pipes & Unix Domain Sockets

-Hauptspeicher (RAM-Disk)

Unix Dateisystem – Terminologie und Kommandos

Was ist ein Unix-Dateisystem?

-Eine Baumstruktur aus benamten Dateien und Verzeichnissen (Wurzel „/“)

-Eine formatierte Festplatten-Partition

-Eine Datenstruktur für Festplatten und Wechselmedien

- UFS, ext2, ext3, ReiserFS
- NFS-Protokoll

-Sonstige Dateisysteme

- ISO9660, FAT, NTFS
- SMB-Protokoll

Wichtige Kommandos	
df	Anzeige des Platzverbrauchs aller Partitionen
du	Platzverbrauch von Dateien/Verzeichnissen
mount	Dateisysteme montieren/anzeigen

Unix Dateisystem – Eigenschaften von Dateien

Eigenschaften von Dateien und Verzeichnissen unter Unix

-(Name)

-Eigentümer und Eigentümer-Gruppe (als numerische UID/GID)

-Zeitstempel (also 32bit Timestamp)

- *access time*
- *modification time*
- *change time*

-Link-Zähler

-Typ

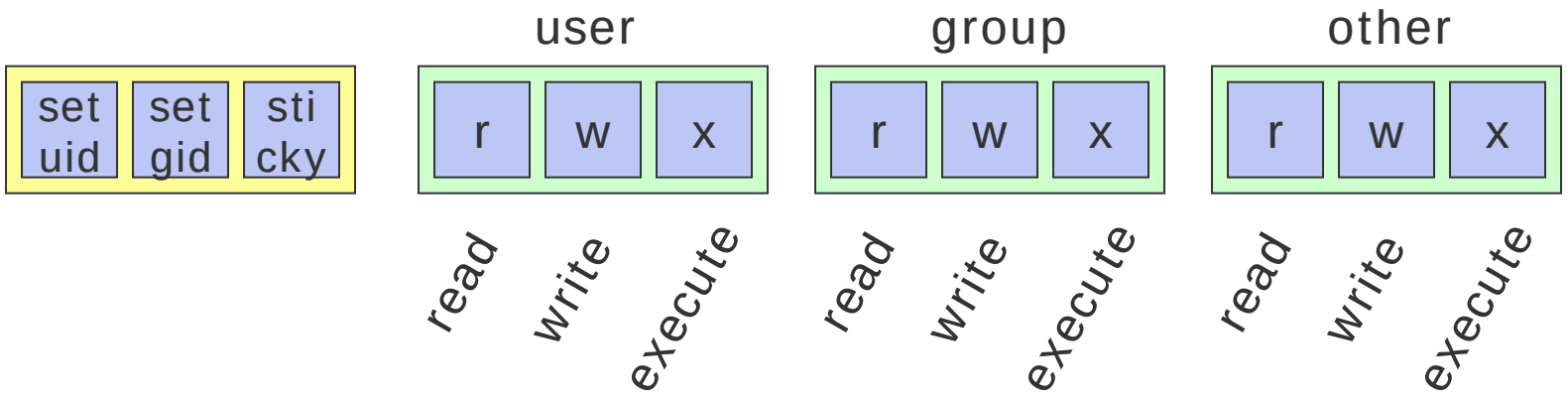
- Datei
- Verzeichnis
- Symbolische Links
- Geräte (Zeichen- und Blockgeräte)
- Unix Domain Sockets
- Pipes

-Zugriffsrechte

Beispiel: Inode des Linux ext2-Dateisystems

```
struct ext2_inode {
  __u16  i_mode;                        /* File mode */
  __u16  i_uid;                         /* Owner ID */
  __u32  i_size;                        /* Size in bytes */
  __u32  i_atime;                       /* Access time */
  __u32  i_ctime;                       /* Creation time */
  __u32  i_mtime;                       /* Modification time */
  __u32  i_dtime;                       /* Deletion Time */
  __u16  i_gid;                         /* Group ID */
  __u16  i_links_count;                 /* Links count */
  __u32  i_blocks;                      /* Blocks count */
  __u32  i_flags;                       /* File flags */
  __u32  i_block [EXT2_N_BLOCKS];       /* Ptrs to blocks */
  __u32  i_version;                     /* File version for NFS */
  __u32  i_file_acl;                    /* File ACL */
  __u32  i_dir_acl;                     /* Directory ACL */
  __u32  i_faddr;                       /* Fragment address */
  __u8   l_i_frag;                      /* Fragment number */
  __u8   l_i_fsize;                     /* Fragment size */
};
```

Unix-Dateisystem – Zugriffsrechte



Oktal-Darstellung

000 110 100 100 = 0644
100 111 111 101 = 4775

„ls –l“

```
-rw-r--r--
-rwsr-xr-x
```

Dateitypen (erstes Zeichen):
- = Datei
d = directory (Verzeichnis)
l = Link (symbolic link)
s = Socket (Unix Domain Socket)
…

	File	Directory
r	Lesen	Listen
w	Schreiben	Anlegen
x	Ausführen	chdir()

Zugriffsrechte: Besonderheiten

-execute (x) bei Verzeichnissen: User kann hineinwechseln

-setuid

- bei Dateien: Ausführen unter der UID des Eigentümers der Datei
- bei Verzeichnissen: neue Dateien gehören dem Verzeichnis-Eigentümer

-setguid

- bei Dateien: Ausführen unter der GID der Eigentümergruppe der Datei
- bei Verzeichnissen: neue Dateien gehören der Eigentümer-Gruppe des Verzeichnisses

-sticky

- bei Dateien: Programm bleibt im Hauptspeicher für weiteren Aufruf (obsolet)
- bei Verzeichnissen: in welt-schreibbaren Verzeichnisse dürfen bei gesetztem Sticky-Bit Dateien nur von Ihren Eigentümern gelöscht werden

-setuid, setgid und sticke wirken nur bei gesetztem x-bit!!

- Sonst: Inkonsistenz wird durch Großbuchstaben dargestellt

Unix-Dateisystem – Zugriffsrechte II

Beispiele

/bin/ls

/tmp

/dev

/etc/shadow

/usr/bin/sudo

/bin/netcat

Weitere Kommandos	
stat	Anzeigen von Datei-Attributen
touch	Zeitstempel von Dateien ändern / leere Datei anlegen, falls diese nicht existiert
ln	Symbolischen Link oder Hardlink anlegen
chown	Ändern des Eigentümers von Dateisystemobjekten
chgrp	Ändern der Gruppe von Dateisystemobjekten
chmod	Ändern von Zugriffsrechten von Dateisystemobjekten

Bash – I/O Umleitung

Jeder Prozess hat normalerweise 3 geöffnete Dateien

-Standardeingabe **stdin** (Filehandle 0)

-Standardausgabe **stdout** (Filehandle 1)

-Standardfehlerausgabe **stderr** (Filehandle 2)

Diese sind für gewöhnlich mit dem Terminal des Benutzers verbunden (also Eingabe durch die Tastatur, Ausgabe auf den Bildschirm), können jedoch auf der Kommandozeile umgeleitet werden:

- „stdin“ kann mit „< Datei“ mit einer Datei verknüpft werden
- „stdout“ kann mit „> Datei“ mit einer Datei verknüpft werden
- „stderr“ kann mit „2> Datei“ mit einer Datei verknüpft werden

Beispiele

```
grep root < /etc/passwd > /tmp/out.txt 2> /tmp/err.txt
ls –l > /tmp/out_and_err.txt 2>&1
find / > /dev/null
```

Aufgabe 2: Erweiterung von superadduser

Problem: ein existierendes Script zum Anlegen von Unix-Usern soll so erweitert werden, dass jeder neue User auch gleichzeitig einen virtuellen Webserver bekommt.

-Lege die Verzeichnisse ~/bin und ~/etc an

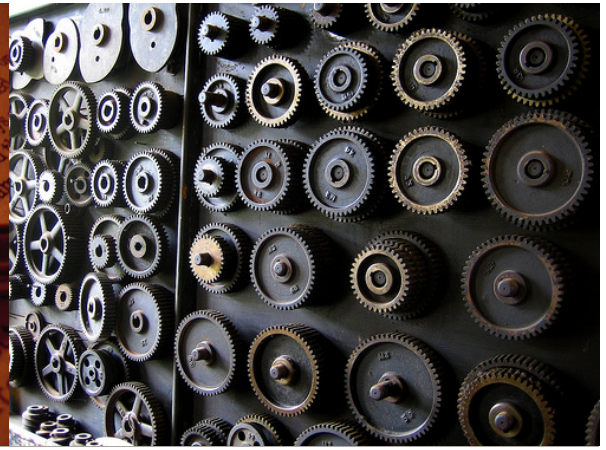
-Kopiere /var/tmp/superadduser nach ~/bin

-Kopiere /etc/passwd nach ~/etc

-Erweitere das Script derart, dass …

- es nicht auf /etc/passwd sondern auf ~/etc/passwd operiert
- die User-Homes nicht unter /home sondern in ~/home liegen
- die privilegierten Kommandos useradd, passwd, chfn und chown nicht verwendet werden (useradd und chfn müssen anders realisiert werden, passwd und chown kann einfach auskommentiert werden)
- zusätzlich für jeden neuen User ein virtueller Host angelegt wird
  - ein htdocs in seinem Home
  - ein VirtualHost in einer neuen Datei unter ~/apache2/etc/sites-available (Portnummer == UID)
  - ein eigenes Log-File

Freie Universität Berlin



Unix Tools

Textverarbeitung mit grep, sed, awk

Reguläre Ausdrücke: Chomsky-Hierarchie

Grammatik	Regeln	Sprachen	Automaten	Abgeschlossenheit
Typ-0	$\alpha \rightarrow \beta$ $\alpha \in V^*NV^*, \beta \in V^*, \alpha \neq \epsilon$	rekursiv aufzählbar	Turingmaschine	KSV*
Typ-1	$\alpha A \beta \rightarrow \alpha \gamma \beta$ $A \in N, \alpha, \beta, \gamma \in V^*, \gamma \neq \varepsilon$ $S \rightarrow \varepsilon$ ist erlaubt, wenn es keine Regel $\alpha \rightarrow \beta S \gamma$ in P gibt.	kontextsensitiv	linear platzbeschränkte nichtdeterministische Turingmaschine	CKSV*
Typ-2	$A \rightarrow \gamma$ $A \in N, \gamma \in V^*$	kontextfrei	nichtdeterministischer Kellerautomat	KV*
Typ-3	$A \rightarrow aB$ (rechtsregulär) oder $A \rightarrow Ba$ (linksregulär) $A \rightarrow a$ $A, B \in N, a \in \Sigma$	regulär	Endlicher Automat	CKSV*

Quelle: wikipedia.de

-REs sind eine Darstellungsform für reguläre Sprachen (WORN)

-verschiedene Ausprägungen

- Basic Regular Expressions (BREs)
- **Extended Regular Expressions (EREs)**
- Perl Compatible Regular Expressions (PCREs)

-können durch endliche Automaten implementiert werden

-werden von vielen Unix-Tools verwendet (sed, awk, grep, perl ...)

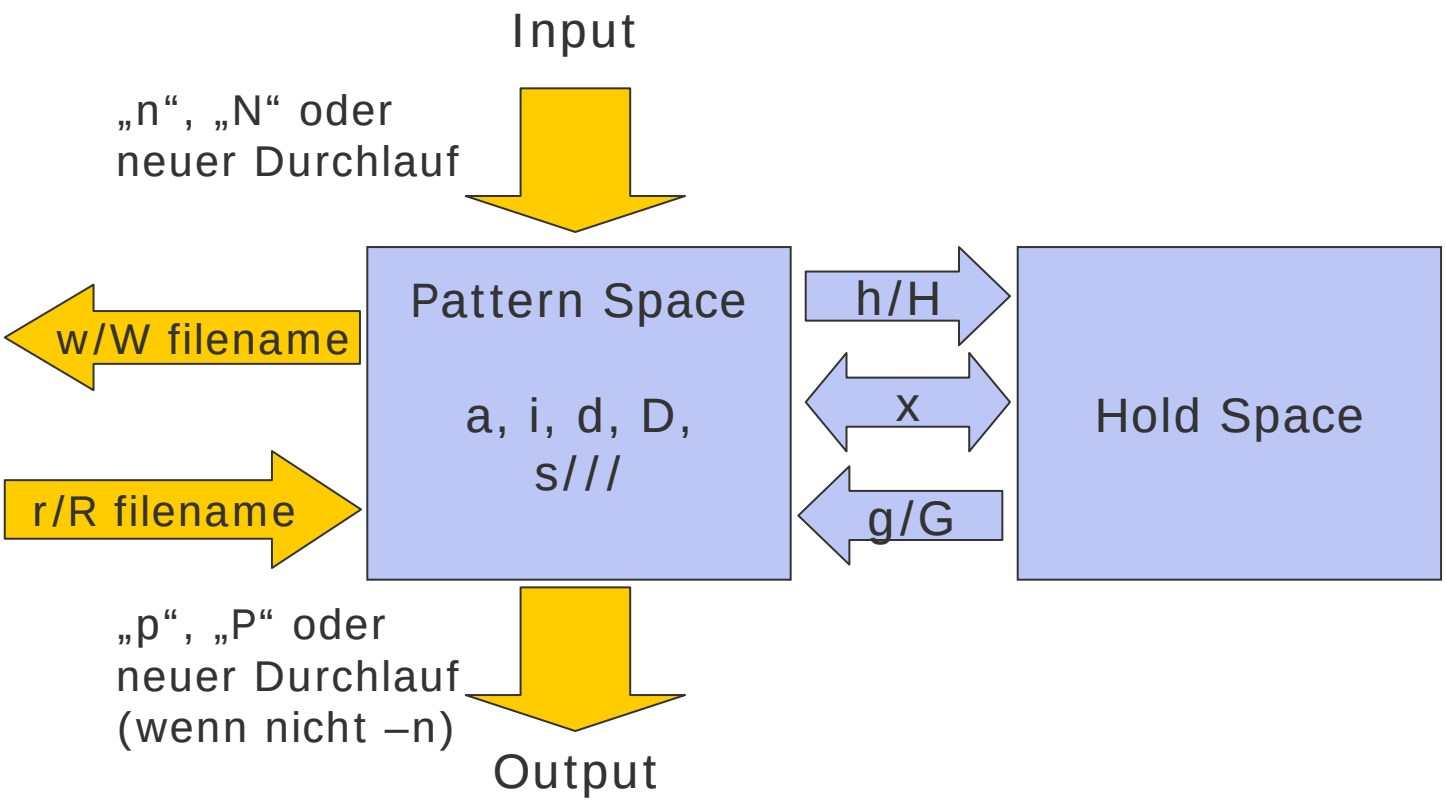
- gibt‘s auch in Java, JavaScript, VBScript, Powershell ...

Reguläre Ausdrücke II

Konzept	**Regulärer Ausdruck**
Literal	„x“ „\.“ (maskieren des Sonderzeichens)
Zeichenmengen (1 aus n)	„.“ (beliebiges Zeichen) „[xyz]“ „[a-zA-Z0-9]“ „[[:alpha:]]“ „[[:digit:]]“ „[[:space:]]“ … (siehe regex(7))
Wiederholung	„(„<regex>“)?“: 0 oder 1mal „(„<regex>“)*“: 0 oder beliebig oft „(„<regex>“)+“: mindestens 1mal
Konkatenation	<regex1><regex2>
Alternativen	<regex1>“|“<regex2>
Zeilenanfang-/-ende	^, $

Reguläre Ausdrücke suchen mit grep

Der Stream Editor „sed“



Der Stream Editor „sed“ (2)

-Struktur eines sed-Scripts (GNU-sed):

sed_script := <statement>*
statement := [condition] (<command_group>|<*command*>)
condition := *empty*|*linenumber*|“/“regex“/“|linerange
linerange := *min*“-“*max (GNU-sed only)*
command_group := „{„ *command*+ „}“

-Kommandos

- Zeile löschen (d), einfügen (i/a)…
- Text ersetzen (s/regex/replacement/)
- Buffer-Handling (x/g/G,h,H)
- Datei einfügen (r/R)

-Autoprint on oder off (-n)

SED: Beispiel

-Reihenfolge der Zeilen vertauschen:

```
#!/usr/bin/sed –nf

1 { h }
1!{ x; H }
${ x ; p }
```

-Textersetzung, z.B. die Interpreter aller Perl-Scripts austauschen

```
find / -name *.pl | while read S; do
        mv $S $S~
        sed –e ‚1 s/^#!.*perl.*/\/usr\/bin\/perl –w/‘ $S~ > $S
done
```

AWK

-Scriptsprache zur Verarbeitung von Text

-Aho, Weinberger, Kernighan

-Verhalten ähnlich wie sed

- Text wird Zeilenweise gelesen, bearbeitet und wieder ausgegeben
- Kommandos-Blöcke mit { }
- Bedingung vor jedem Kommandoblock: RegEx ... BEGIN/END

-Unterschiede zu sed

- Die Syntax der Scripte ist an C angelehnt
- Es wird eine **Tabellenstruktur** vorausgesetzt: jede Zeile wird automatisch an einem Trennzeichen (FS) aufgespalten und den Variablen **$1, $2** ... zugewiesen

awk-Beispiel

Statistik über die verwendeten Shells von Usern

```
ypcat passwd | shell_stats.awk | sort -n
```

shell_stats.awk:

```
#!/usr/bin/awk -f

BEGIN{
        FS=„:“
}

{
        ## Arrays sind assoziativ!
        count[$NF]++
}

END{
        for( v in count )  printf( "%3i %s\n", count[v], v )
}
```

Textverarbeitung

Wichtige Kommandos	
cut	Schneidet Zeichen oder Spalten aus Textzeilen
sort	Sortiert Text-Dateien nach Spalten
(e)grep	Sucht Zeilen in Textdateien, die auf einen Ausdruck passen
sed	Stream Editor – Scriptgesteuerter Texteditor für Textströme
awk	Scriptsprache für die Textverarbeitung, insbesondere für tabellarische Daten
m4	???

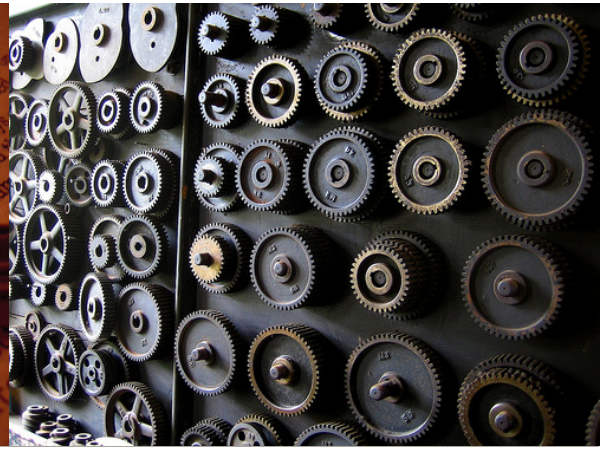
Aufgabe: Apache Logfile Analyse

Schreibe ein awk-Script, das die Logfiles des Apache liest und die Zugriffe pro Objekt und Tag zählt. Die Ausgabe sollte eine Tabelle der folgenden Form sein:

```
+-----+------+------+
+ URI | Date | Hits |
+-----+------+------+
|…                   |
+--------------------+
```

Damit wenigstens ein paar Einträge im Log stehen, kann mit siege ein Stress-Test durchgeführt werden.

Freie Universität Berlin



Die Unix Prozesshierarchie

Definition, Lebenszyklus, Kommandos, Beispiele

Bash – Hintergrund-Prozesse & Job-Kontrolle

Hintergrund-Prozesse startet man mit „&“ am Zeilenende:

find / -name *.jpg > /tmp/find_out 2>&1 &

(sinnvollerweise mit umgeleiteter IO!)

Job-Kontrolle

- Vordergrund-Auftrag stoppen: Strg-Z
- Liste der Jobs: jobs
- Job in den Vordergrund holen: fg [<num>]
- Job im Hintergrund weiterlaufen lassen: bg [<num>]

Beispiele

grep im Hintergrund starten

vi unterbrechen

xeyes in den Hintergrund

Unix Prozesse – Definition

Ein Unix Prozess ist ein **Programm während der Ausführung**.

Attribute eines Prozesses

-Systemweit eindeutige numerische ID (PID)

-ID des Eltern-Prozesses (PPID, die Prozess-Hierarchie ist ein Baum!)

-Effektiver Eigentümer (EUID), effektive Gruppe (EGID)*

-Offene Dateien (inklusive der Lese-Position) *

-Umgebungsvariable (Name-Wert-Paare) *

-Aktuelles Arbeitsverzeichnis *

-Code-Segment und Programmzähler *

-* Bei der Erzeugung geerbte Informationen

Beendet sich ein Prozess, liefert er einen numerischen **Status** an seinen Elter-Prozess, der diesen mit **wait()** abfragen muss.

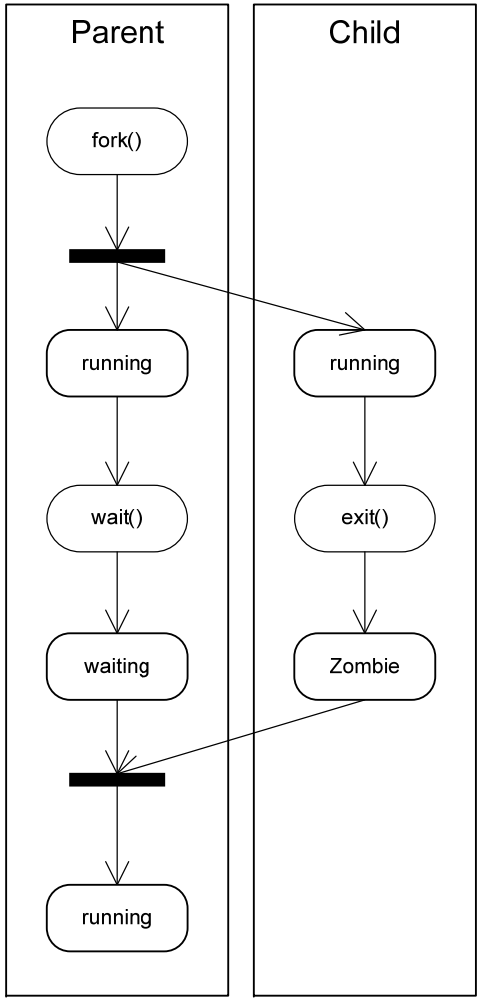
Init-Prozess

-erster nach dem Systemstart erzeugter Prozess (PID 1)

-Erzeugt alle weiteren Prozesse, die für Benutzeranmeldungen erforderlich sind

-Dient als Eltern-Prozess für verwaiste Prozesse

Unix-Prozesse - Lebenszyklus



	Parent	Child
wait() vor exit()	Blockiert oder läuft weiter	Läuft weiter
exit() vor wait()	Läuft weiter (erhält SIGCHLD)	Wird zum Zombie bis zum wait()
Unbe-handeltes Signal an Parent	Verschwindet	Wird auch gekillt
Parent beendet sich	Verschwindet	Bekommt **init** als Parent-Prozess (der auf alle Prozesse wartet)

Unix-Prozesse – Erzeugung und Kommandos

fork(2) erzeugt nur einen Kindprozess als Kopie:

```
If ( ( pid = fork() ) > 0 ) {
        // Parent-Prozess, PID in pid
} else {
        // Kind-Prozess
}
```

Um ein anderes Programm auszuführen, muss dieses mittels **exec(3)** in den neuen Kindprozess geladen werden.

Kommandos	
ps	Prozessliste ausgeben
pstree	Prozessliste als Baum ausgeben
top	Prozesse in Echtzeit, sortiert nach Ressourcenberbrauch
kill	Signal an Prozess senden (Prozess beenden)
export <var> (Shell Builtin)	Macht <var> zu einer Umgebungsvariable

Unix-Prozesse – Beispiele

Xeyes stoppen und weiterlaufen lassen

/proc

Warum muss „cd“ als Shell-Builtin implementiert sein, „pwd“ jedoch nicht?

Zombie erzeugen:

```
perl -e 'if ((my $pid = fork() ) > 0) { print $pid."\n"; while(1){} } else { exit }'
```

Prozess an init hängen:

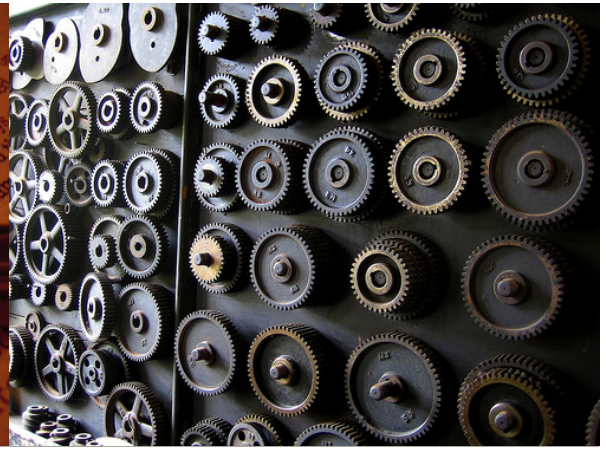
```
perl -e 'if ((my $pid = fork() ) == 0) { while(1){} } else { print $pid.“\n“ ; exit }'
```

Aufgabe: Parallelisierung der Log-Analyse

Schreibe ein Shell-Script, welches ein Apache-Log in n gleichgroße Teile splittet und n awk-Prozesse mit der Log-Analyse darauf ausführt und anschließend die Ergebnisse ausfummiert. Wie schnell ist diese Vorgehen im Vergleich zu nur einem Prozess?

Kommandos: wc(1), split(1), wait (Shell-Builtin)

Freie Universität Berlin



Sonstige Schweinereien

ssh, tar, nc, cron/at, CGI-Scripting

Sonstiges

-ssh

-tar

-tar durch ssh

-X-Forwarding mit ssh

-ssh-Tunneling (z.B. rdesktop)

-CGI-Scripting

-make