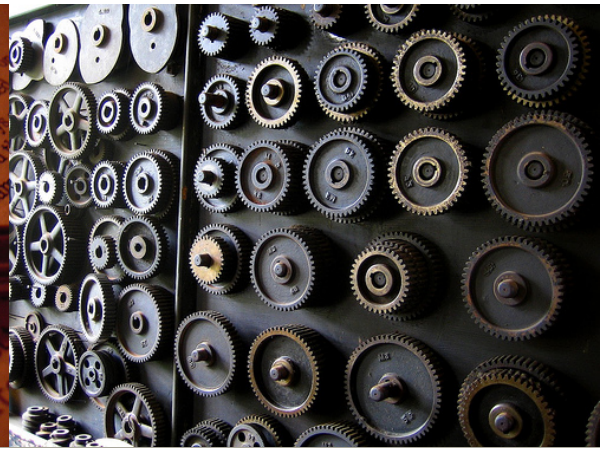
Freie Universität Berlin



Systemverwaltung Sommersemester 2008
am Fachbereich Mathematik und Informatik

Scripting und Automatisierung

Daniel Bößwetter boesswet@inf.fu-berlin.de

Juli/August 2008

Freie Universität Berlin

Motivation / Inhalt

-Warum sollte man den Systembetrieb automatisieren?

- Steigerung der Qualität: Maschinen machen weniger Fehler als Menschen
  - Schonmal versucht, 1000 User zwischen Unix und Windows synchron zu halten?
- Spart Arbeit: der Administrator kann sich um sinnvollere Aufgaben kümmern
- Zuverlässigkeit: automatisierte Aufgaben können auch zu allen (un-) möglichen Zeiten ausgeführt werden, oder wenn die Umstände dies erfordern (z.B. Platte voll)

-Inhalt dieser Veranstaltung

- Schwerpunkt auf Scripting
- Unix
  - Grundlagen
  - Shell-Scripting
  - Dateien & Prozesse
  - Text-Verarbeitung
  - Netzwerk und Sonstiges
- Windows
  - BatchFiles
  - PowerShell
- Wenn die Zeit reicht: Blick über den Tellerrand

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

Organisatorisches

Termine

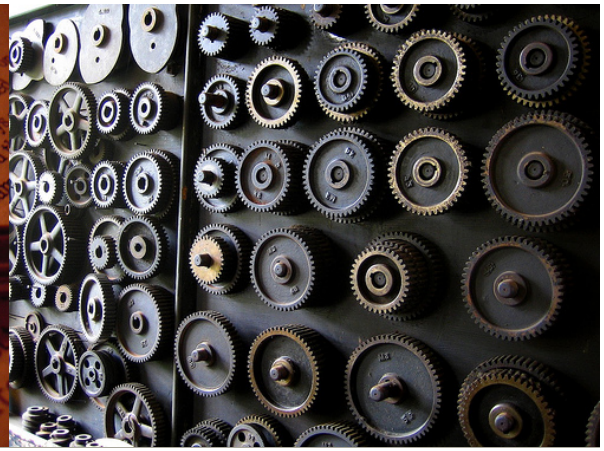
Do. 31.07	Fr. 01.08	WE	Mo. 04.08	Di. 05.08	Mi. 06.08	Do. 07.08
Unix	Unix		Unix	Unix	Windows	Windows

Nachmittags 13-15

Pro Block 15 min Pause.

Im Bioinformatik-Poolraum

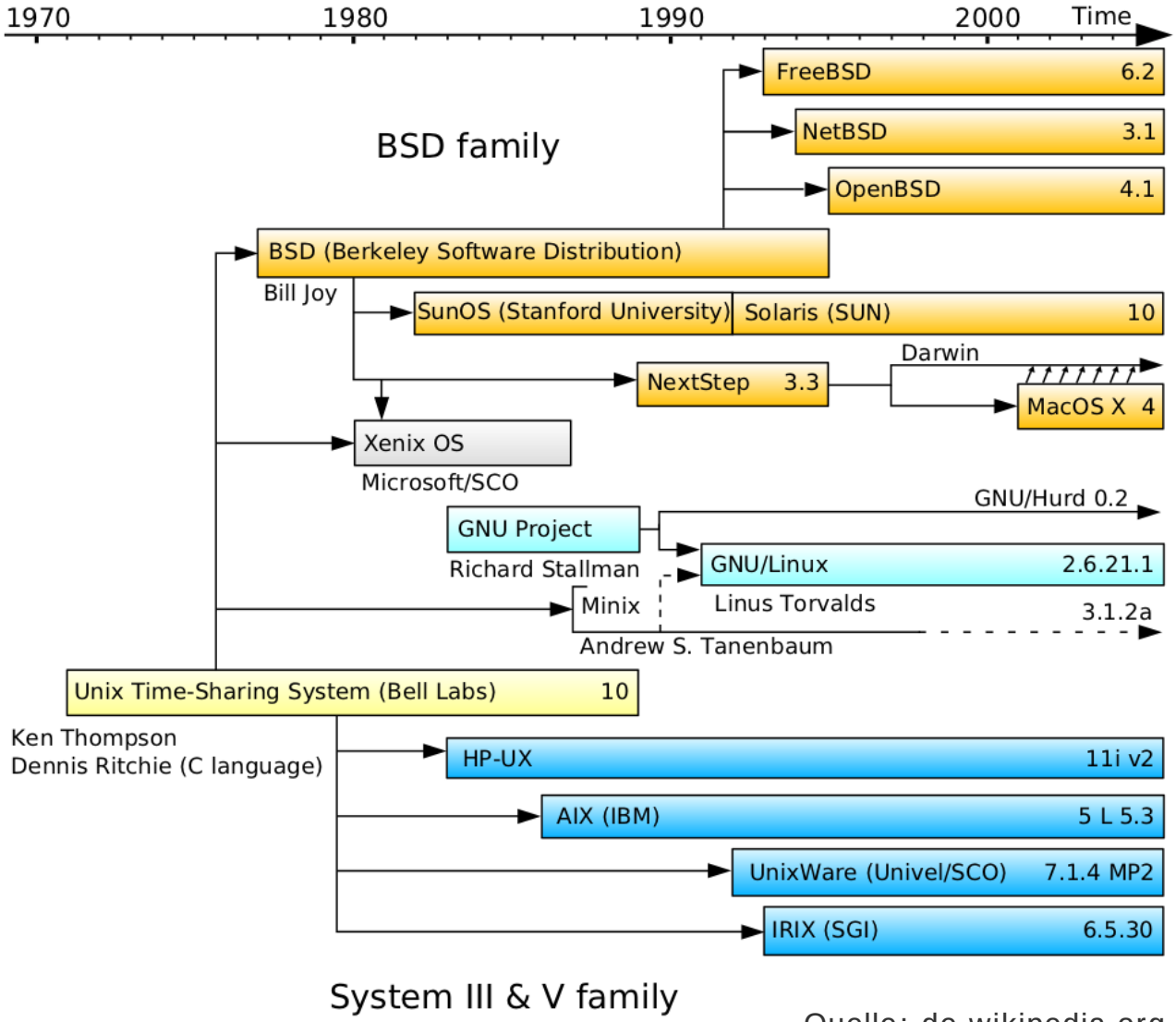
Freie Universität Berlin



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

-whoami, id, who, finger

-/etc/passwd, /etc/group, root

Ken Thompson, Dennis Ritchie

Linus Torvalds, Andrew S. Tannenbaum

Unix Literatur

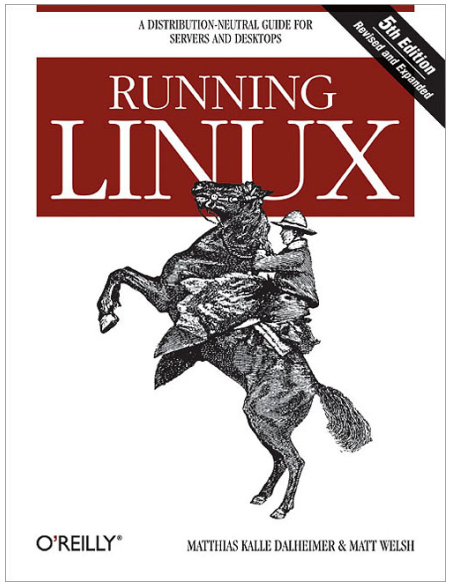
Suche nach „Unix“ auf amazon.de liefert 898 Ergebnisse …

Eine gute Einführung in Linux ist z.B.

Running Linux, Fifth Edition

BY MATTHIAS KALLE DALHEIMER, MATT WELSH
FIFTH EDITION DECEMBER 2005

O‘REILLY



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

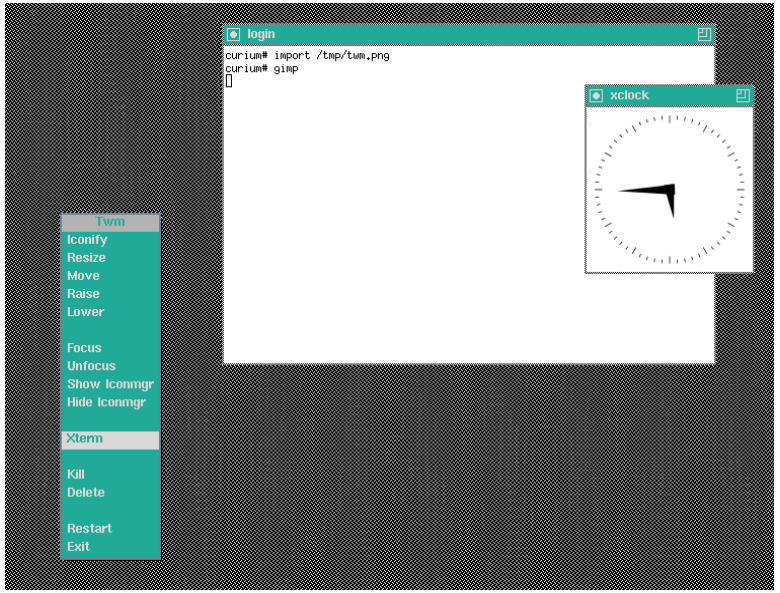
Unix – Basics

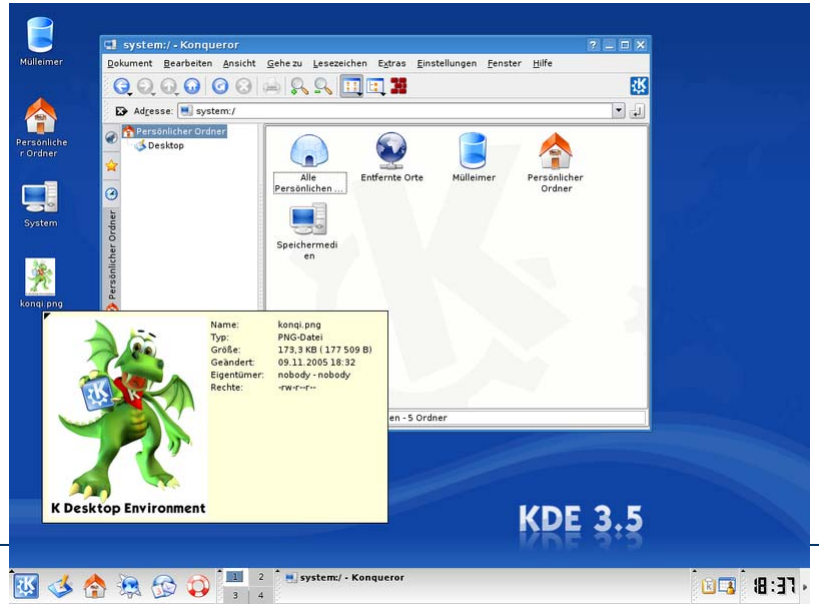
-**Grafische Oberflächen**
- X11
- KDE
- Gnome

-**Shells** als Kommandozeilen-Interpreter
- Interaktiv
- Scripting

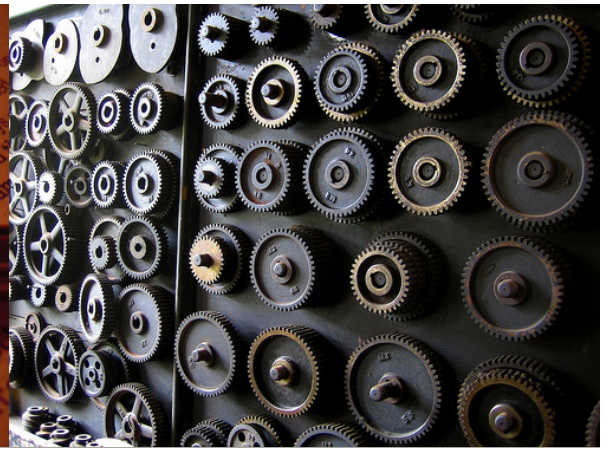
-**Kommandos**
- Meistens ausführbare Programme
- Manchmal Shell-intern
- Optionen

-**Manualseiten** („man-pages“)
- für die meisten Kommandos, Systemaufrufe, Dateiformate, Funktionen vorhanden …
- „Wow, das Unix-Handbuch!“ (Garth in *Waynes World*)





Freie Universität Berlin



Arbeiten mit Unix-Shells

Einführung, interaktives Arbeiten, Manualseiten, Variable, Expansion, Job-Kontrolle

Unix-Shells

Ursprünglich als Schnittstelle zwischen Benutzer und Kern („Schale“)

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

-Einrichten/Starten der Bash

Unix-Kommandos

Was sind Unix-Kommandos?

-Einfache Befehle

- Shell-intern (Job-Kontrolle, Variable, …)
- 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

Wichtige Kommandos	
man	Manualseite lesen
pwd cd <dir>	Aktuelles Verzeichnis ausgeben/wechseln
ls	Verzeichnisinhalt auflisten
more, less, cat	Text-Dateien lesen, konkatenieren
mkdir, rmdir	Leeres Verzeichnis anlegen/löschen
cp, mv, rm	Datei kopieren/verschieben (umbenennen)/löschen
file	Dateityp-Erkennung
id, whoami	Eigene Identität
env	Umgebungsvariable
hostname	Rechnername
uname	OS-Informationen

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**

- **Control-C bricht laufende Kommandos ab**

Übung: Navigation im Dateisystem

1. Starte ein Terminal mit einer Bash darin.
2. Um welche Betriebssystem-Version handelt es sich?
3. Lege in Deinem Home ein Unterverzeichnis namens „admin_kurs“ an und wechsle in das neue Verzeichnis hinein.
4. Lege dort zwei Unterverzeichnisse „a“ und „b“ an, wobei „b“ ein Unterverzeichnis „c“ haben soll (das geht mit einem Kommando!)
5. Wechsle ins Verzeichnis „a“.
6. Kopiere die Datei /boot/vmlinuz ins lokale Verzeichnis
   - Liste den Inhalt des Verzeichnisses auf.
   - Wem gehört diese Datei und wann wurde sie erstellt?
   - Wem gehört das Original und wann wurde das erstellt?
   - Um was für einen Dateityp handelt es sich bei vmlinuz?
7. Lege mit touch(1) eine Datei namens „.xyz“ (mit einem Punkt am Anfang!) an.
   - Liste den Inhalt des Verzeichnisses auf.
   - Wie groß ist die neue Datei?
8. Wechsle wieder ins „admin_kurs“-Verzeichnis und
   - prüfe, dass Du im richtigen Verzeichnis bist!!!
   - lösche „a“ mit nur einem rm-Aufruf.
   - lösche „b“ mit nur einem rmdir-Aufruf.

Bash – Variable und Expansion

- **Variable**
  - Shell-Variable: nur in der Shell verwendbar
  - Environment-Variable: werden an Kindprozesse vererbt (Beispiel: PATH)

- **Expansion *vor* der Ausführung von Kommandos**
  - Variablen-Expansion ($var bzw ${}, $$, $?, ...)
  - Shell-Globs *, ?
  - Tilden-Expansion ~
  - Ausdrücke $[]
  - Leerzeichen Maskieren mit „“ oder _
  - Maskierung der Expansion ‚...‘ oder \$
  - Kommando-Substitution $() oder Backticks `...`

Bash – Beispiele zur Expansion

-Variablen-Expansion ($var bzw ${}, $$, $?, ...)

-Shell-Globs *, ?
    *
    /bin/l*
    /bin/l?.d

-Tilden-Expansion ~

-Ausdrücke $[1+1], $[SHELL==bash]

-Quotes
    Dateinamen mit Spaces

-Maskierung mit \ oder „...‘

-Kommando-Substitution
    Logfile-Name mit `date`

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 – 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
```

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

Bash – komplexe Kommandos

Sequentielle Ausführung:

$ date ; sleep 10 ; date

Logisches UND / ODER der Return-Codes (Lazy-Evaluation)

$ test –f /tmp/xxx || touch /tmp/xxx

$ test –f /tmp/xxx && rm /tmp/xxx

Pipelines (Ausgabe eines Kommandos als Eingabe für das nächste):

$ ls –l / | grep bin

Wichtige Kommandos	
find	Dateien suchen
xargs	Kommandos auf mehrere Dateien ausführen
tee	Pipeline mitprotokollieren
grep	Sucht in Textdateien
touch	Datei anlegen/Datums-stempel anpassen
head, tail (-f)	Dateianfang/-ende
sleep	Sekundenschlaf
read	Zeile lesen

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

Wichtige Kommandos	
seq	Aufzählung
test []	Vergleiche und Datei-Tests
expr	Ausdrücke auswerten

Bash – Beispiel

Wir wollen in einem Verzeichnis alle .c-Dateien in „.c~“ umkopieren, um diese ggfs. später wieder herstellen zu können.

```
find . –name „*.c“ | while read name ; do mv $name $name~ ; done
```

Oder (nicht rekursiv)

```
for name in *.c ; do mv $name $name~ ; done
```

Übung: Expansion und xargs

Lege in Deinem „admin_kurs“-Verzeichnis mit einer Schleife 9000 Dateien an, die jeweils „datei_“ gefolgt von einer laufenden Nummer heissen.

Lösche alle Dateien wieder.

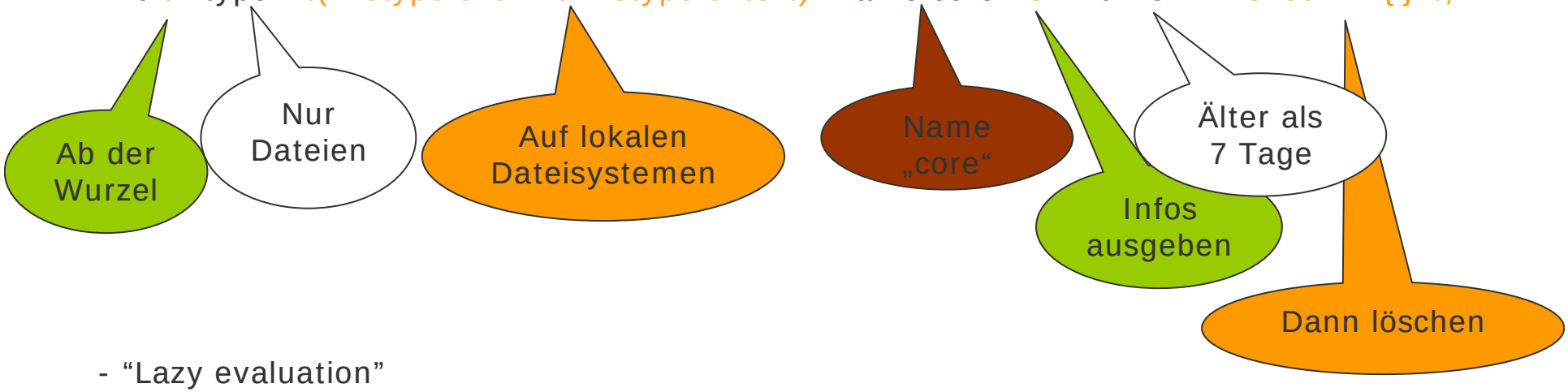
Tipp: das xargs-Kommando wird zum löschen gebraucht. Warum?

Beispiel: find

Problem:
finde alle core-Files auf lokalen Dateisystemen. Gib Pfad, Größe und Eigentümer aus und wenn die Datei älter als 7 Tage ist, lösche sie.

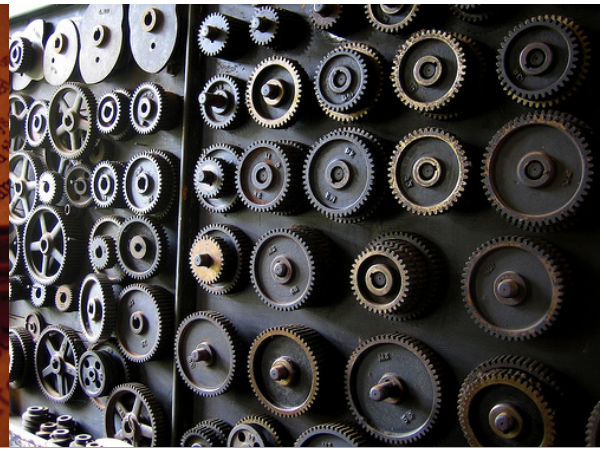
Lösung:

```
find / -type f \( -fstype ext2 –o –fstype ext3 \) -name core -ls -mtime +7 -exec rm {} \;
```



- “Lazy evaluation”

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)

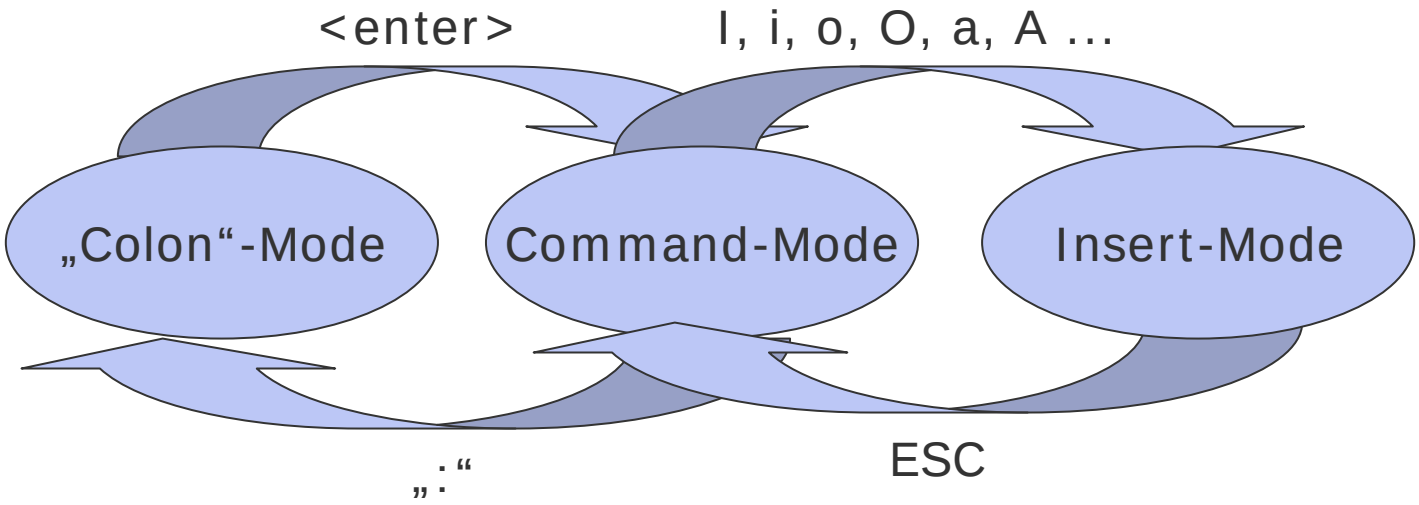
Wenn das Script nicht im Suchpfad ($PATH) liegt: ausführen mit Pfadangabe!

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

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)

VIM: Makros

Beispiel: TPC-H Script?

Aufgabe: Shell-Script

Unter Unix ist es nicht möglich, einmal gelöschte Dateien wieder herzustellen. In grafischen Oberflächen gibt es daher die (von Windows bekannte) “Mülleimer”-Metapher, d.h. löschen bewirkt nur ein verschieben in einen “Trash”-Folder, aus dem bis zur endgültigen Löschung Dateien wieder rekonstruiert werden können.

Wir wollen ein solches Konstrukt für die Shell bereitstellen. Dazu brauchen wir folgende Scripten:

-„myrm_rm.sh“, welches als Ersatz für rm(1) dienen kann (also auch die Kommandozeilen-Parameter versteht!) aber die Dateien z.B. nach ~/.trash verschiebt.

-„myrm_list.sh“, dass den Inhalt des Mülleimers auflistet

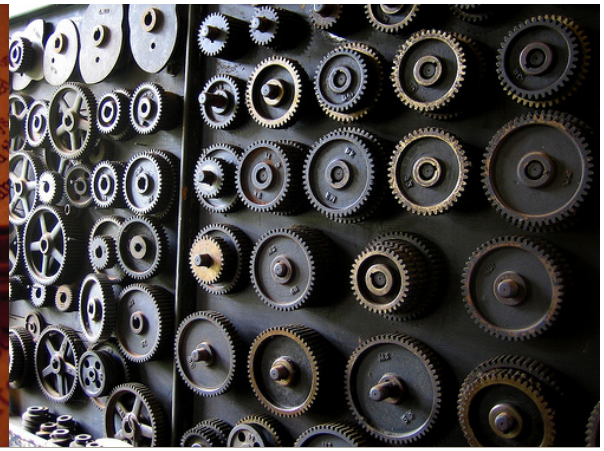
-„myrm_undelete.sh“, dass eine Datei wiederherstellt

-„myrm_dispose.sh“, welches den Mülleimer entleert.

Es kann natürlich sein, dass mehrere Dateien mit dem gleichen absoluten Pfad nacheinander gelöscht werden! Symlinks, Partitionen und Rechte können erstmal ignoriert werden.

Tipp: mktemp(1), stat(1), fgrep(1)

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

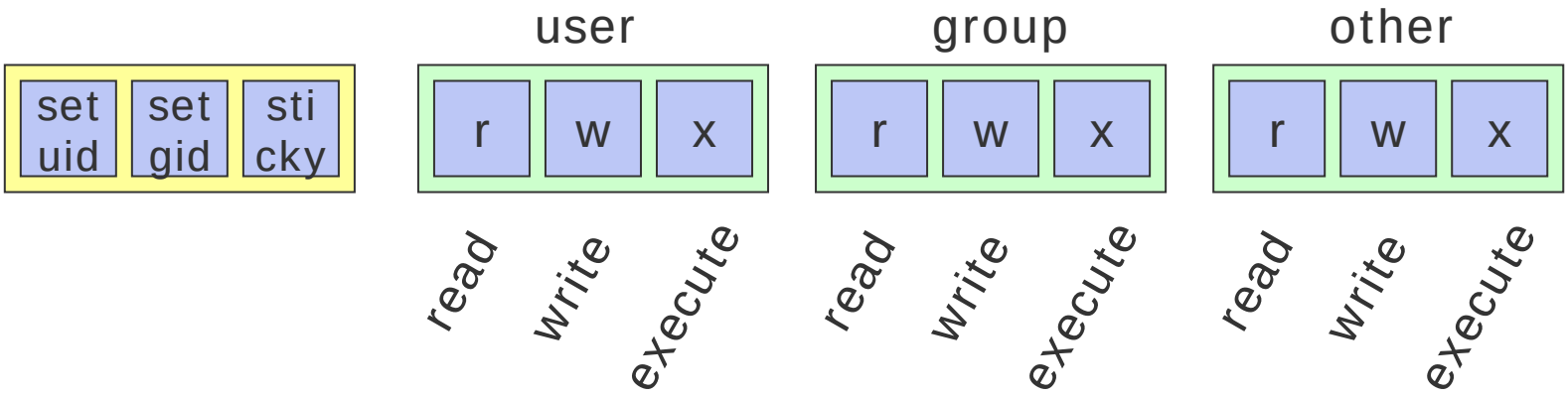
- Datei
- Verzeichnis
- Symbolische Links
- Geräte (Zeichen- und Blockgeräte)
- Unix Domain Sockets
- Pipes

-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 **„ls –l“**

000 110 100 100 = 0644 `-rw-r--r--`

100 111 111 101 = 4775 `-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

Aufgabe

Schreibe ein Script, das für alle lokalen Partitionen prüft, ob noch mindestens 90% der Inodes frei sind. Andernfalls soll eine Warnung ausgegeben und mit einem entsprechenden Return-Code terminiert werden.

Tipp: das Shell-Builtin „set“ könnte helfen …

Aufgabe: Erweiterung des myrm-Scripts

Erweitere das myrm-Script so, dass Dateien/Verzeichnisse, die

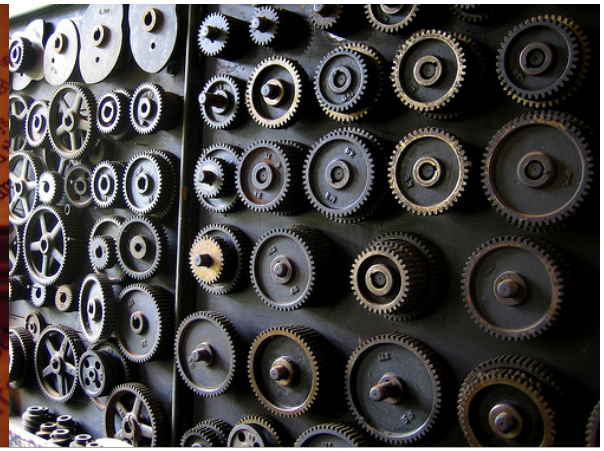
-mehr als ein definiertes Maximum an Platz verbrauchen (z.B. 1GB) oder

-nicht auf der gleichen Partition wie der Mülleimer liegen

direkt gelöscht und verschoben werden

Kommandos: df –P, du

Freie Universität Berlin



Die Unix Prozesshierarchie

Definition, Lebenszyklus, Kommandos, Beispiele

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.

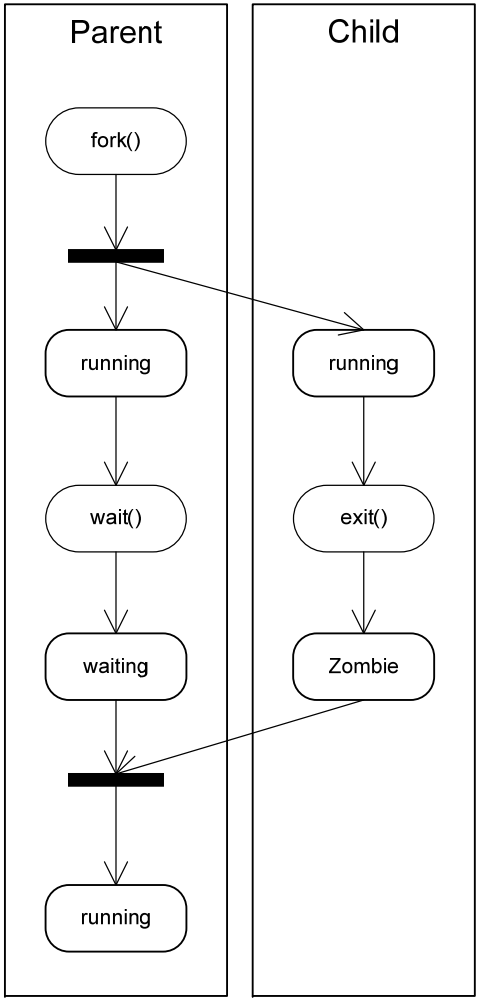
Init-Prozess

-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?

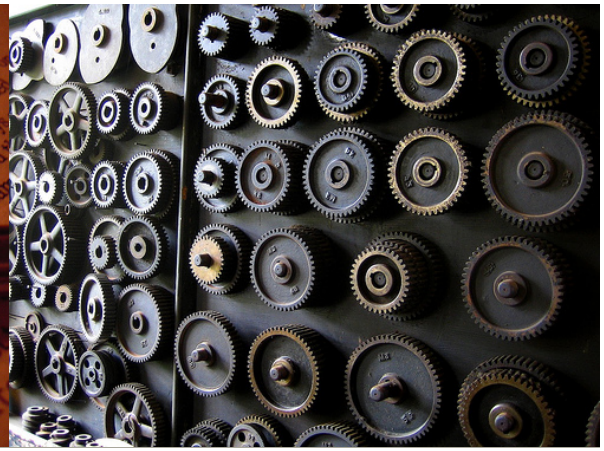
Zombie erzeugen:

```
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 }'
```

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 ...)

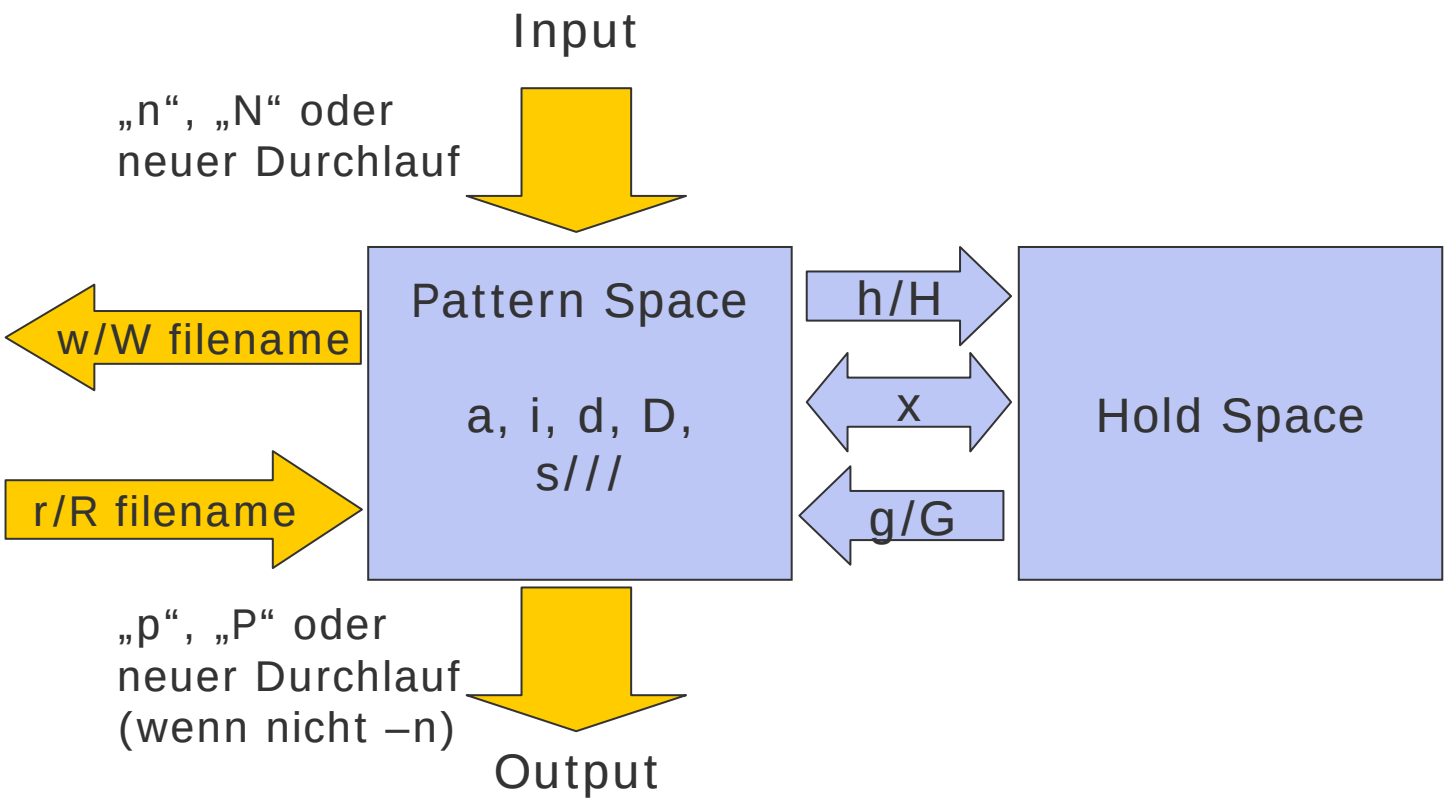
- gibt‘s auch in Java, JavaScript, VBScript, Powershell ...

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

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	???

Textverarbeitung: Aufgabe

Der Postfix-Mailserver kennt eine Datei namens „virtuals“, in der (beliebige) E-Mail-Adressen auf Benutzerkonten abgebildet werden. Diese hat das Format:

email@domain username

Schreiben Sie ein Skript, welches eine virtuals-Datei für Postfix aufbaut, wobei diese aus 2 Teilen besteht:

1. Einem fixen Präfix, der in der Datei selbst steht und vom Dateianfang bis zu einer Zeile mit dem Inhalt „#---END“. Diese Zeilen müssen erst gerettet werden, bevor die Datei überschrieben wird, weil hier z.B. Administrative Adressen wie postmaster und webmaster definiert sind.
2. Einer Zeile für jeden Benutzer-Account mit einer UID größer 1000 aus der /etc/passwd (sollte irgendwo konfigurierbar sein) mit dem Format vorname.nachname, wobei der Name aus dem Kommentarfeld extrahiert wird. Beachten Sie, dass Email-Adressen keine Leerzeichen oder Kommas enthalten dürfen!

Textverarbeitung: Aufgabe (2)

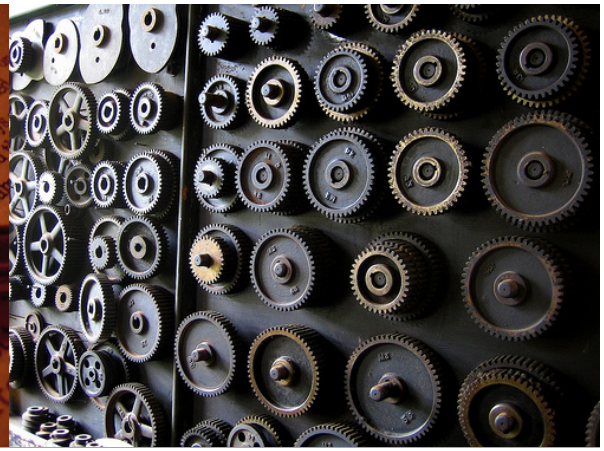
3 Schritte:

1. Kopiere virtuals nach virtuals~ (z.B. mit cp(1))
2. Schreibe alle Zeilen aus virtuals~ bis zur #---END nach virtuals (die dabei überschrieben wird) (z.B. mit sed(1))
3. Lese passwd und erzeuge für alle Zeilen mit uid > 1000 eine Zeile, die an virtuals angehängt wird (z.B. mit awk(1))

Beispiel-Dateien (virtual und passwd) unter

http://page.mi.fu-berlin.de/boesswet/

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