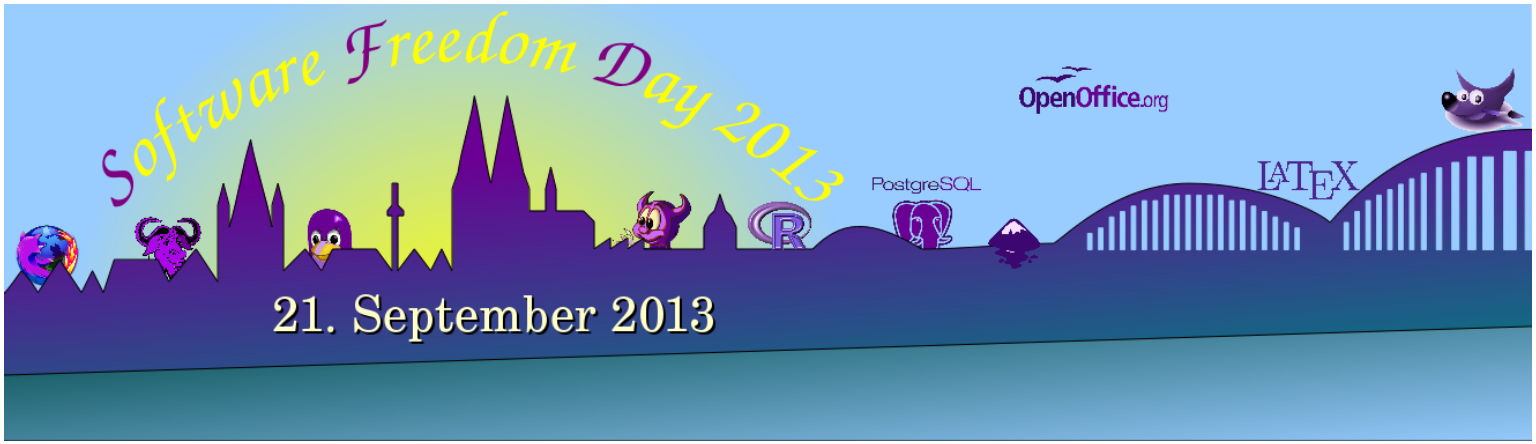
Die Kommandozeile – der vergessene Riese

Volker Ossenkopf

1. Physikalisches Institut der Universität zu Köln



Überblick

- Wo geht's zur Kommandozeile?
- Kommandos in Bäumen
  - man
  - ls, cd, mkdir, rmdir
- Die Shell
  - Die bash
  - Variablen
  - Umleitungen
  - Shell-Programmierung
- Suchen und Finden
- Prozesskontrolle
- Das Ende

Wo geht's zur Kommandozeile?

Die Textkonsole:

- Umschalten aus dem X-Window-System mit Alt-Strg-F1 ... F4
- Zurück ins X mit Alt-Strg-F7 (manchmal Alt-Strg-F5)

Terminal-Fenster: xterm (das Original), rxvt, aterm, eterm, gnome-terminal, katerm, tterm, ...

Wichtigste Tastenkombinationen:

- Shift-PageUp, Shift-PageDown – Blättern
- linke Maustaste – Markieren/Kopieren (Beachte: Anzahl der Mausklicks)
- rechte Maustaste – Markierungsende
- mittlere Maustaste – Einfügen

Das wichtigste Kommando

man

```
man(1)                    Manual Hilfsprogramme                    man(1)

NAME
       man - Programm zum Einsehen der Online-Manuale

SYNTAX
       man [-acdhwutZV] [[Abschnitt] Seite ...] ...

BESCHREIBUNG
       man ist der Manualbrowser des Systems. Jedes Argument Seite ist nor-
       malerweise  der  Name eines Programmes oder einer Funktion. Gefunden
       und angezeigt wird die Manualseite, die  auf  alle  Argumente  paßt.

BEISPIELE
       man ls
           zeigt die Manualseite für das Programm ls an.
```

```
OPTIONEN
       -M Pfad, --manpath=Pfad
              Ermöglicht  es, einen alternativen Manualpfad anzugeben.
       -k, --apropos
              Diese Option ist das Äquivalent zu apropos.  Es  werden  die
              Kurzbeschreibungen zu allen Manualseiten nach dem angegebenen
              Stichwort  durchsucht.

SIEHE AUCH
       mandb(8), manpath(1), manpath(5), apropos(1), whatis(1),  catman(8),
       less(1),  nroff(1),  troff(1),  groff(1)

FEHLER
       Die Option -t funktioniert nur, wenn  ein  troff-ähnliches  Programm
       installiert ist.
```

Allgemeine Kommandosyntax: `Kommando [-Optionen] Argumente`

- Kommandos: z.B. man, ls, openoffice.org, cdrecord, shutdown
- Optionen: z.B. -n, -v –verbose, –with-gnu
  Bedeutung verschieden je nach Kommando
- Argumente: z.B. Dateinamen, Zahlen, Zeichenketten
- Trennung jeweils durch Leerzeichen

Kommandos in Bäumen

Alle Programme und Dateien eines Unix-Systems sind in einer Baumstruktur organisiert.

Bewegen in der Baumstruktur:

`cd` - change directory

Besondere Verzeichnisse:
- `/` = Wurzelverzeichnis des Gesamtsystems
- `~` = Heimatverzeichnis des Benutzers
- `..` = übergeordnetes Verzeichnis
- `.` = aktuelles Verzeichnis

Anzeigen der enthaltenen Dateien:

`ls` - list

Wichtige Optionen:
- `-l` = “long listing” = erweiterte Ausgabe
- `-a` = “all files” = auch versteckte Dateien

Kommandos in Bäumen

Alle Programme und Dateien eines Unix-Systems sind in einer Baumstruktur organisiert.

Verzeichnisse Erstellen und Löschen:

`mkdir` - make directory - erstelle Verzeichnis

`rmdir` - remove directory - lösche Verzeichnis

Die wichtigsten Kommandos finden sich unter `/bin`:

```
bash, cat, chgrp, chmod, chown, cp, date, df, echo,
false, find, grep, kill, ln, login, ls, mkdir,
mknod, more, mount, mv, ps, pwd, rm, rmdir, sed,
sh, sleep, stty, touch, true, umount, uname
```

(Grundgerüst jedes Unix-Systems)

Weitere Kommandos

Jede ausführbare Datei ist ein Kommando.

☞ Sie muss nur gefunden werden.

Wie finde ich das passende Kommando?

- Built-in Namenserweiterung ⟨TAB⟩
- apropos
- “Siehe auch” Kapitel der manpages

Gefunden werden nur Kommandos im Pfad ☞ PATH-Variable.
z.B.

```
> echo $PATH
/home/ossk/bin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

Ein ganz besonderes Programm - die Shell

Die bash ist die Linux-Standardshell.

Alle Kommandozeileneingaben werden von der Shell interpretiert und verarbeitet. Die meisten hier diskutierten Eigenschaften der “Kommandozeile” sind Eigenschaften der bash.

Die bash

- ersetzt Teile von Kommandos und Argumenten
- erlaubt Kommandoverknüpfungen
- führt interne Befehle aus
- speichert und verarbeitet Variablen

Zeichenersetzungen – Wildcards

*	beliebige Zeichenzahl
?	genau ein Zeichen
~	das Home-Verzeichnis des Nutzers

Das wichtigste bash-Kommando

help

```
GNU bash, version 4.2.25(1)-release (x86_64-pc-linux-gnu)
These shell commands are defined internally.  Type ‘help’ to see this list.
Type ‘help name’ to find out more about the function ‘name’.

 job_spec [&]                                history [-c] [-d offset] [n]      >
 (( expression ))                            if COMMANDS; then COMMANDS;       >
 . filename [arguments]                      jobs [-lnprs] [jobspec ...]
 :                                           kill [-s sigspec ] pid |  ..      >
 [ arg... ]                                  let arg [arg ...]
 [[ expression ]]                            local [option] name[=value] ...
 alias [-p] [name[=value] ... ]              logout [n]
 bg [job_spec ...]                           mapfile [-n count] [-O origin]    >
 bind [-lpvsPVS] [-m keymap] >               popd [-n] [+N | -N]
 break [n]                                   printf [-v var] format [arguments]
 builtin [shell-builtin [arg ...]]           pushd [-n] [+N | -N | dir]
 caller [expr]                               pwd [-LP]
 case WORD in [PATTERN COMMANDS ;;] >        read [-ers] [-a array] [-d delim] >
 cd [-L|[-P [-e]]] [dir]                     readarray [-n count] [-O origin]  >
 command [-pVv] command [arg ...]            readonly [-aAf] [name[=value] ...]
```

Arbeiten mit Variablen

- Zugriff auf Variablen mit `$`
- Anzeige einzelner Variablen mit `echo`
- Anzeige aller Variablen mit `set`
- Setzen von Variablen mit `Variable=Wert`
- Weitergabe von Variablen an aufgerufene Programme mit `export`

Beispiele:

- `echo $PATH`
- `PS1="Was jetzt?:  "`

Besondere Variablen:

`$0` ... `$9`	Kommando und Argumente
`$@`	alle Argumente
`$#`	Anzahl der Argumente
`$?`	Rückgabestatus
`$PATH`	Pfad für Kommandos
`$LANG`	Spracheinstellung

Arbeiten mit Variablen

Die Shell kann mit Variablen "rechnen".
Einfache Verarbeitung des Inhaltes von Variablen

Operationen:

`$[ Zahl1 +/- Zahl2 ]` Addition

`String{String1,String2}` Klammererweiterung zur Duplizierung

`${Variable:Anfang:Länge}` Teilstring extrahieren

`${Variable#Vari}` Zeichenentfernung am Anfang

`${Variable%able}` Zeichenentfernung am Ende

Beispiel:

- `V1=$COLUMNS; V2=2; V3=Unsinn`
- `V4=$[$V1-$V2]`
- `V5=$V4+$V3`
- `V6=${V5%sinn}`
- `echo ${V6:0:4}`
- `mkdir /usr/local/share/{foo,bar}`

Quotes

Es gibt drei verschiedene Anführungszeichen:

’String’ keinerlei Interpretation durch die Shell

"Liste" alle Variablen werden durch die Shell ersetzt

‘Kommando‘ es wird die Ausgabe des Kommandos verwendet

Beispiele

- `PS1=‘hostname‘’ zu Diensten:’`
- `NOW=‘date +%M‘`
- `echo ’Time to talk: ’$[60 -$NOW]’ minutes.’`

Spezialitäten der Shell - Umleitungen und “Pipes”

Jeder Prozess besitzt (mindestens) 3 Ein-/Ausgabekanäle:

0 – Standardeingabe

1 – Standardausgabe

2 – Standardfehlerausgabe

Diese lassen sich zu Dateien und “Pipes” umlenken:

`<Datei` oder **0** `<Datei`	Eingabeumleitung
`>Datei` oder **1** `>Datei`	Standardausgabeumleitung
2 `>Datei`	Fehlerausgabeumleitung
2 > &1	Fehlerausgabe auf die Standardausgabe lenken
`>>Datei`	Standardausgabe an Datei anhängen
`| Kommando`	Standardausgabe an Kommando durchreichen

Pipes

Die Mächtigkeit der Kommandozeilenarbeit beruht zum großen Teil auf der Verküpfung von Kommandos mit Pipes:

- `Kommando1 Argument | Kommando2 | Kommando3 | Kommando 4`

Nutzung zum Stream-Editieren:
`sed` filtert und bearbeitet den Datenstrom in nahezu beliebiger Weise.

Beispiel:

```
sed 's/typischer Vertiiper/typischer Vertipper/'
                                        Originaldatei >NeuerText
```

Um die Ausgabe eines Kommandos gleichzeitig zu speichern und am Bildschirm zu sehen, gibt es das nützliche Kommando `tee`.

Beispiel:

```
strace Kommando 2>&1 | tee protokoldatei
```

Programmierung mit der Shell

- Bedingte Programmausführung mit `&&` bzw. `||`
- Schleifen mit `for` Variable `in` Liste`;` `do` Kommandos`;` `done`
- Bedingte Verzweigung mit `if` Kommando`;` `then` Kommandos`;` `else` Kommandos`;` `fi`

Nützliches Kommando für Bedingungen:
`test` oder `[ ]`

Beispiele:

- `test -z "$LD_LIBRARYPATH"`
- `[ $USER = 'ossk' ]`
- `[ $[ ${Mein_Counter} % 10] -eq 1 ]`

Programmierung mit der Shell

Eigene Programme ausführen:

- Textdatei anlegen
- Kommandos hineinschreiben
- Argumente durch $1-$9 oder $@ ersetzen und verarbeiten
  mehr Argumente nach und nach mit `shift` abarbeiten
- Hashbang zur Shell in der ersten Zeile einfügen:
  `#!/bin/bash`
- Datei als ausführbar deklarieren:
  `chmod +x Datei`

Beispiel:

```
cat rename-all

#!/bin/bash
for i in *
do
mv -i "$i" `echo $i | sed -e s/"$1"/"$2"/g`
done
```

Suchen und Finden

Dateinamen:

`locate` sucht nach Dateien mit bestimmten Zeichenketten im Namen

- Benutzt interne Datenbank, die täglich upgedated wird
- Sehr schnell, aber nicht immer aktuell

Dateieigenschaften:

`find` sucht Dateien nach Name, Größe, Erstellungsdatum, Typ, ...

- `-name` = Namensbestandteile, z.B. '*.mp3'
- `-type` = Dateityp, z.B. Verzeichnis oder reguläre Datei
- `-size` = Dateigröße, z.B. größer als 10MB
- `-mtime` = Aktualisierungsdatum, z.B. jünger als 1 Monat
- `-exec` = Kommando, das für die Datei ausgeführt werden soll, z.B. ls -l {} \;
- `-and / -not` = logische Verknüpfung für Bedingungen oben

Beispiel: `find . -size +1M -exec ls -l {} \; |sed -e 's/ */ /g' |cut -d' ' -f '5-9'|sort -g -r`

Suchen und Finden

Text in Dateien:

`grep` durchsucht Dateien nach regulären Ausdrücken

Reguläre Ausdrücke:

`.`	= beliebiges Zeichen
`*`	= beliebige Wiederholung
`^`	= Zeilenanfang
`$`	= Zeilenende
`[0123] [A-Z]`	= Zeichen aus einer Gruppe
`{3}`	= feste Wiederholung

Häufige Optionen:

- `-H` = Ausgabe des Dateinamens
- `-i` = Groß- und Kleinschreibung ignorieren
- `-e` = Beginn des Suchmusters, wenn mehrere angegeben werden
- `-c` = Nur Anzahl der Treffer ausgeben

Plattenplatz:

`du` zeigt den Plattenplatz an, den bestimmte Verzeichnisse belegen.
`df` zeigt den Plattenplatz auf den verschiedenen Geräten an.

Prozessverwaltung

Prozesse, die von der Kommandozeile gestartet wurden, lassen sich dort direkt kontrollieren:

- **Strg-C** = abbrechen
- **Strg-Z** = stoppen
- **Strg-D** = Eingabe endgültig beendet

Hintergrundverarbeitung:
Wird ein Kommando mit `&` abgeschlossen, wird es im Hintergrund ausgeführt und blockiert das Terminal nicht länger.

Zurückgegeben wird eine Prozessnummer (Variable $!).

Gestoppte Prozesse oder Hintergrundprozesse werden mit

`fg` im Vordergrund

`bg` im Hintergrund

fortgesetzt.

Prozessverwaltung

Eine Übersicht aller laufenden Prozesse erhält man mit:

`ps`	zeigt alle laufenden Prozesse in verschiedenen Formaten
`top`	zeigt die Systemresourcen und die CPU-intensiven Prozesse an

Alle Prozesse lassen sich beenden mit:

`kill`	nach Prozessnummer
`killall`	nach Prozessnamen

In beiden `kill`-Kommandos kann man die Prozesse mit verschieden“schweren” Signalen beenden:

- `kill -2` = freundliche Anfrage
- `kill -9` = finaler Todesstoß

Zusammenfassung

- Nur mit der Kommandozeile lässt sich ein Linux/Unix-System voll beherrschen.
- Die Unix-Grundbefehle bieten Werkzeuge für außerordentlich viele Aufgaben.
- Mit `grep, sed, sort` lassen sich komplexe Aufgaben schnell lösen.
- Mit `grep` und `find` wird Ordnung zu halten beinahe überflüssig  .
- Fehlersuche und Prozesskontrolle funktioniert nur auf der Kommandozeile universell.

Das Ende:

☞ `exit`