Shell 101 and Unixy Utils

Alexander Gehrke

May 8, 2019

Worum geht's eigentlich?

Shell a.k.a. Kommandozeile / Terminal / CLI ...
das Programm, das im Terminal Befehle verarbeitet

"Unixy" Unix-Prinzip: kleine Programme, die man kombinieren kann

Shell 101

Shells

diverse Shells zur Auswahl

- bekannteste: bash
- beliebte Alternative: zsh
- kleinster gemeinsamer Nenner: POSIX sh
- diverse Shells mit anderer Syntax

Im Basics-Teil dieses Vortrags: nur Syntax, die in allen POSIX-Shells gehen sollte.

Befehle aufrufen

```
$ ls
doc.pdf
file1.txt
file2.txt
fileX.txt
script.sh
```

Befehle aufrufen

```
$ ls -l
total 0
-rw-r--r-- 1 crater2150 crater2150 0 Apr 20 21:11 doc.pdf
-rw-r--r-- 1 crater2150 crater2150 0 Apr 20 21:11 file1.txt
-rw-r--r-- 1 crater2150 crater2150 0 Apr 20 21:11 file2.txt
-rw-r--r-- 1 crater2150 crater2150 0 Apr 21 13:21 fileX.txt
-rwxr-xr-x 1 crater2150 crater2150 0 Apr 20 21:11 script.sh
```

Globbing

Globbing = Dateinamen nach Muster suchen

```
$ ls file*
file1.txt
file2.txt
fileX.txt
$ ls *.pdf
doc.pdf
$ ls file[0-9]*   # keine Regexes!
file1.txt
file2.txt
```

Globbing — rekursiv

```
$ mkdir -p foo/bar/baz foo/bar/qux
$ touch foo/bar/baz/baz.txt foo/bar/qux/qux.txt
$ ls */*
baz
qux
```

Globbing — rekursiv

```
$ ls **/*
doc.pdf
file1.txt
file2.txt
fileX.txt
foo/bar/baz/baz.txt
foo/bar/qux/qux.txt
script.sh

foo:
bar
```

```
foo/bar:
baz
qux

foo/bar/baz:
baz.txt

foo/bar/qux:
qux.txt
```

Globbing — rekursiv

```
$ echo **/*
doc.pdf file1.txt file2.txt fileX.txt foo foo/bar foo/bar/baz
↪  foo/bar/baz/baz.txt foo/bar/qux foo/bar/qux/qux.txt script.sh
```

Variablen

- Variablen in Befehlen werden ersetzt, signalisiert durch Dollar: `$VAR_NAME`.
- Setzen einer Variable mit `VAR_NAME="wert"` — hier kein Dollar!
- Ersetzung auch in Strings mit `"double quotes"`, nicht in `'single quotes'`

```
$ myvar="World"
$ echo "Hello, $myvar."
Hello, World.
```

Schleifen — while

Übliche Schleifenarten: For und While.

While-Syntax:

```
$ while condition; do commands; done
#example:
$ while ! ping -c 1 -q heise.de; do
    echo "Waiting for internet"
    sleep 5
  done; wget https://example.com/file.zip
```

Bedingung ist beliebiges Programm. Ausschlaggebend ist Exit Code des Programms.

Schleifen — for

For-Syntax:

```
$ for name in elements; do commands; done
#example
$ for i in *.jpg; do convert $i $i.bmp; done
```

- ”name” ist beliebiger Variablenname
- ein Schleifendurchlauf pro Element, Element wird der Variable zugewiesen
- Elemente sind häufig Globbing-Muster, aber auch Programm-Output oder Text möglich. Separiert wird in letzteren Fällen immer an Whitespace (im Zweifelsfall escapen)

Unix-Prinzip: IO

Unix-Prinzip = kombinieren von Programmen. Auf der Shell meist über Standardein- und ausgabe.

Dafür gibt es verschiedene Möglichkeiten:

- Ausgabe eines Programms in Befehl einfügen
- Aus Dateien lesen und in diese schreiben
- Ausgabe eines Programms als Eingabe des nächsten Programms nutzen

IO: Command Substitution

Möglichkeit 1: Ausgabe vom Programm in anderen Befehl einfügen.

```
$ for i in $(seq 1 10); do
    echo $i
  done
```

`seq` gibt Zahlenreihen aus (z.B. für indexbasierte Schleifen)

IO: File redirection

Möglichkeit 2: Dateien lesen und schreiben

- `< file` Aus Datei lesen
- `> file` In Datei schreiben (ersetzt Inhalt)
- `>> file` An Datei anhängen

```
$ grep '^###' < ../shell.md > headlines.txt
$ cat headlines.txt
### Shells
### Befehle aufrufen
### Globbing
### Input / Output
```

IO: Piping

Möglichkeit 3: Ausgabe eines Programms an ein anderes Programm weitergeben

```
$ my_great_log_program | grep 'Error' | less
```

- `my_great_log_program` gibt Logs aus
- `grep` sucht nach ”Error”, behält nur Zeilen mit Match
- `less` ist ein Pager: macht Ausgabe scroll- und durchsuchbar

If-then-else

```
$ if condition; then
    something;
  elif other-condition; then
    something-else
  else
    something-completely-different
  fi
```

Conditions

Bedingung in `if` und `while` sind Programme. Einige Programme sind speziell für die Verwendung als Bedingung gedacht.

- `true` und `false` geben immer 0 bzw. 1 zurück
- `test` ist Standardbefehl für Bedingungen. Beispiele
  - Stringvergleich: `test $var = str`
  - Zahlenvergleich: `test $var -gt 5` (`-gt` = greater than)
  - Prüfen ob Datei existiert: `test -e file.txt`

  Kann auch als eckige Klammern geschrieben werden: `[ $var -gt 5 ]`

Bei modernen Shells üblicherweise built-in, teilweise erweiterte oder geänderte Syntax.

Short conditions

Ähnlich wie bei anderen Sprachen können Bedingungen mit `&&` und `||` verknüpft werden.

Da Bedingung = Programm, kann man das zur Fehlerbehandlung oder für verkürzte ifs nutzen:

```
$ run_some_job || \
    printf "Subject: Job failed\n\nFailed at $(date)" \
    | sendmail admin@example.com


$ compile_program && run_program
```

Unixy utils

Outline

Disclaimer

Es gibt viele Programme, die unter ”basic” fallen können (175 Programme schreibt der POSIX Standard für eine konforme shell vor). Hier also eine kleine Auswahl

Die Zeit reicht auch für die wenigen nicht, um sie alle genau durchzugehen. Wer zu einem Programm mehr wissen will, einfach melden.

Man

Auf den folgenden Folien wird nur noch selten stehen, wie man die Programme genau benutzt. Praktischerweise liefern sie alle Manuals mit.

```
$ man $befehl

# optional section, wenn mehrere Manpages mit dem Namen
$ man 1 $befehl
```

Text verarbeiten

grep	Suchen nach Regexen
head/tail	Nur gewisse Anzahl Zeilen vom Anfang / Ende behalten
sort / uniq	Sortieren, doppelte Zeilen filtern, Vorkommen zählen
wc	Steht für word count, zählt aber auch Zeilen oder Zeichen

Dateisystem

cd	change directory, Arbeitsverzeichnis wechseln
ls	Dateien auflisten
rm / rmdir	Löschen von Dateien / Verzeichnissen
touch / mkdir	Erstellen von leeren Dateien / Verzeichnissen
chmod / chown	Berechtigung und Besitzer ändern

Dateisystem – Fortsetzung

ln	Dateisystem-Links erstellen
find	Nach Dateien und Verzeichnissen suchen
file	Infos über Datei (z.B. Typ, Encoding, etc.)
diff	Unterschiede zwischen Dateien anzeigen

Programme, die einen eigenen Talk füllen würden

awk Die Manpage beschreibt es als ”pattern scanning and processing language”. Ideal zum Verarbeiten tabellarischer Daten (z.B. CSV). Verarbeitet Input zeilen- und spaltenweise.

sed Steht für Stream Editor. Auch eine eigene Scriptsprache mit Fokus auf Textverarbeitung. Häufigster Anwendungsfall: Suchen und ersetzen mit Regex.

Just for fun: häufigste Aufrufe

```
awk '{print $1}' $HISTFILE \  # mit awk nur erste spalte der history
  | sort | uniq -c \          # sortieren und dann gleiche zeilen
  ↪  zählen
  | sort -nr \                # numerisch sortieren, absteigend
  ↪  (reverse)
  | head -n 10                # nur erste 10 Zeilen
```

Just for fun: häufigste Aufrufe

```
8245 ll    # Alias für `ls -l`
6672 vim
5825 git
4352 cd
3607 gst   # Alias für `git status`
2537 fg    # Prozess aus Hintergrund vorholen
1966 ag    # Textsuche, sehen wir gleich noch
1372 rm
1296 ssh   # Remote Shell
1258 gap   # Alias für `git add --patch`
```

Also unter Beachtung der Aliases ist eigentlich Git der häufigste Befehl :-)

Weniger bekannte Utilities

Moreutils

Utilities, die zum POSIX-Standard gehören könnten, wenn damals jemand dran gedacht hätte.

chronic	Programmoutput unterdrücken, außer es failed (exitcode nicht 0)
combine	Boolsche Operationen für zwei Dateien (z.B. alle Zeilen die in A vorkommen, aber nicht in B)
ts	jede Zeile mit Timestamp versehen a.k.a. Logfiles für Arme
pee	wie eine Pipe, aber leitet Output an mehrere Befehle
sponge	Braucht man, wenn man eine Datei überschreiben will, die man gleichzeitig als Input in der gleichen Kommandozeile benutzt
vidir	Verzeichnisse mit Texteditor bearbeiten
vipe	Pipe mit Texteditor bearbeiten
...	

Suchen für Ungeduldige und Tippfaule — Text

Wir haben schon `grep` kennengelernt. Damit kann man auch ganze Verzeichnisbäume durchsuchen:

```
$ time grep -r class
...
#viel output
...
grep -r class  0.30s user 0.26s system 43% cpu 1.302 total
```

Suchen für Ungeduldige und Tippfaule — Text

Gleiche Suche, gleicher Ordner, aber mit `ag`

```
$ time ag class
...
# weniger output, dafür bunt
...
ag class  0.05s user 0.07s system 186% cpu 0.061 total
                      Faktor 20 schneller ^^^^^
```

Außerdem: ignoriert Dinge aus Ignorefiles, Suche in bestimmtem Dateityp, allgemein angenehmeres Interface.

Wer unbedingt etwas in einer bestimmen Programmiersprache geschriebenes will oder sehr performancebewusst ist: `ack`, `rg`, `pt`, `ucg`, …

Suchen für Ungeduldige und Tippfaule — Dateien

Posix-Standard: `find`. Z.B. Suche nach allen PDFs im aktuellen Ordner:

```
$ find . -iname '*.pdf'
```

Oder allen Dateien mit foo im Namen:

```
$ find . -iname '*foo*'
```

`find` kann noch viel mehr, aber für häufigste Anwendungsfälle dadurch umständliche Syntax.

Suchen für Ungeduldige und Tippfaule — Dateien

Für Alltagsgebrauch: `fd`. Äquivalente Befehle zu eben:

```
$ fd -e pdf
$ fd foo
```

Bonus Features: ignoriert per Default versteckte und durch ignorefiles ausgeschlossene Dateien. Mehr Farbe. Schneller, selbst ohne alle Ignoreoptionen.

TODO

fzf, exa, jq, rsync, pass, pv, tmux, thefuck...

htop, mutt (+ offlineimap), vim, khal/khard (+vdirsyncer), mpv, mpd, ledger, ranger, ncdu...

xorg interop: xclip, dmenu/rofi, dragon, screen-message...

~~TODO~~ Live Demo

fzf, exa, jq, rsync, pass, pv, tmux, thefuck…

htop, mutt (+ offlineimap), vim, khal/khard (+vdirsyncer), mpv, mpd, ledger, ranger, ncdu…

xorg interop: xclip, dmenu/rofi, dragon, screen-message…