Bash und reguläre Ausdrücke, Jana Cavojska
Mathematik und Informatik
Institut für Informatik

Freie Universität Berlin

Einführung in Bash und reguläre Ausdrücke

Was ist die Bash?

- **Skriptsprache** (eine simple Programmiersprache)

- **Interpreter** für die Skriptsprache "Bash"

- das, was am anderen Ende eines Terminal Emulators steht

- die ausführbare Binärdatei **/bin/bash**

Wozu ist sie gut?

- um coole Linux-Tools aufzurufen

- Interagieren mit den gerufenen Programmen ("interaktive Shell")

- Verbinden mehrerer Programme über Pipes

- Schreiben von simplen Skripten oder komplizierteren Programmen

Linux-Tools

ls

- steht für „list“

ls	zeigt Inhalt eines Verzeichnisses an
ls -l	zeigt Details zu jeder Datei / zu jedem Verzeichnis an
ls -a	zeigt auch versteckte Dateien an
ls -la	kombiniert die Funktionalität von ls -a und ls -l

cd

„change directory“
- wechselt das aktuelle Verzeichnis

cd „/home/jane/meine Dokumente“ wechselt in das genannte Verzeichnis
cd .. wechselt in das Parent-Verzeichnis
cd . wechselt in das aktuelle Verzeichnis, tut also nichts

pwd, cp, mv, echo, cat,

pwd	„print working directory“ (zeigt das aktuelle Verzeichnis an)
cp <datei1> <datei2>	kopiert datei1 nach datei2 („copy“)
mv <datei1> <datei2>	verschiebt (benennt um) datei1 nach datei2 („move“)
echo "some string"	printet "some string" in die Konsole
cat datei.txt	printet den Inhalt der Datei datei.txt in die Konsole

 # dies ist ein Kommentar, weil es hinter einem „#“ steht

find

- findet Dateien nach einer Reihe von Kriterien.

Beispiel:

Finde rekursiv alle Dateien im aktuellen Verzeichnis, die „abc“ irgendwo in Namen stehen haben, keine Verzeichnisse sind, und zum letzten Mal innerhalb der letzten 24 Stunden verändert wurden:

```
find . -iname "*abc*" -type f -mtime -1
```

grep

- ermöglicht das Durchsuchen von Dateien nach ihrem Inhalt

Beispiel:

Printe alle Zeilen der Datei „file.txt“, die das Wort „Haskell“ enthalten:

```
grep „Haskell“ file.txt
```

Printe alle Zeilen in allen Dateien (rekursiv) in allen Unterverzeichnissen des aktuellen Verzeichnisses, die das Wort „Haskell“ enthalten:

```
grep -r „Haskell“ ./
```

Man pages

- bilden die offizielle Dokumentation aller installierten Linux-Tools
- offline verfügbar auf jedem Linux-System
- man page für den grep-Befehl aufrufen:

man grep

- Info zur Organisation der man pages:

man man

- sind auch im Internet verfügbar unter

http://linux.die.net/

Variablen in Bash

Bash kennt keine Datentypen, alles sind Strings.

```
a="a bc"        # ein String
a="123"         # noch ein String
a=123           # eine Zah.... just kidding, auch ein String

#  Die Ausgabe eines Befehls in eine Variable speichern:
a=$(grep "some string" "/path/to/some file")
#  oder, veraltet:
a=`grep "some string" "/path/to/some file"`

#  Den Inhalt der Variable a in die Konsole printen:
echo "$a"

#  Wichtig: keine Leerzeichen bei Wertzuweisungen!
```

Exit status

Exit status eines Befehls überprüfen

echo $?

- wenn der exit status 0 ist -> alles ok
- wenn es etwas anderes ist: in der man page nachschauen

Bash-Skripte schreiben

Jedes Bash Skript muss das hier als erste Zeile enthalten:

```
#!/bin/bash
```

und mit dem Befehl

```
chmod a+x meinSkript.sh
```

o.Ä. ausführbar gemacht werden. Und die Endung .sh haben.

Der Aufruf des Skriptes aus dem aktuellen Verzeichnis sieht dann so aus:

```
./meinSkript.sh
```

Zahlen

```
a=2

((a = a + 3))

Echo $a   # gibt 5 zurück

# Dieses Konstrukt unterstützt leider nur ganze Zahlen.
```

Fallunterscheidungen

Mit Zahlen:

```
a=5

if ((a < 6))
then
   echo "a ist kleiner als 6"
else
   echo "ne, ist nicht kleiner"
fi
```

oder:

```
if ((a < 6)); then echo "a ist kleiner als 6"; else echo "ne, ist
nicht kleiner"; fi
```

Fallunterscheidungen nach Exit Status

```
if grep -q "aa" datei
then
   echo "gefunden"
   cat datei
else
   echo "nicht gefunden"
fi
```

Mit Fließkommazahlen rechnen

- kann Bash von sich aus nicht. Daher: das bc-Tool verwenden:

```
echo "1.2 + 0.2" | bc -l
```

bc in einer Fallunterscheidung benutzen:

```
if [ $(echo "1.2 > 0.2" | bc -l) == 1 ]        # oder == 0
then
   echo "ja"
else
   echo "nein"
fi
```

Datei Zeile für Zeile lesen

```
i=1
while read line
do
   echo "Zeile $i: $line"
   (( i = i + 1 ))
done < "/Pfad/zur/Datei.txt"
```

Alle Dateien in einem Verzeichnis bearbeiten

```
for file in "/tmp/test dir/"*
do
   echo "old filename: $file"
   mv "$file" "${file}neu"
   echo "new filename: ${file}neu"
done
```

I/O-Streams

Jedes Programm in Linux hat 3 mit ihm assoziierte I/O-Streams:

- **0 (stdin, standard input)**
- **1 (stdout, standard output)**
- **2 (stderr, standard error)**

0, 1, 2 sind die file descriptors dieser Streams.

Umleitung der I/O-Streams

In eine Datei schreiben

Umleitung des stdout von echo in die datei.txt, die dadurch erzeugt / überschrieben wird:

```
echo "abc" > datei.txt
```

Umleitung des stdout von echo und Anhängen an datei.txt:

```
echo "abc" >> datei.txt
```

Umleitung des stderr von rm nach /dev/null:

```
rm file.txt 2>/dev/null
```

Umleitung von stdout nach /dev/null und Umleitung von stderr nach stdout

```
ls somedir > /dev/null 2>&1
```

Umleitung der I/O-Streams

Von einer Datei lesen

Datei /tmp/test als Input für grep akzeptieren:

```
grep "ab" < /tmp/test
```

Von /tmp/test mit grep lesen und nach /tmp/test2 mit grep schreiben:

```
grep "ab" < /tmp/test > /tmp/test2
```

Umleitung der I/O-Streams

Pipes

Die "<", ">" Operatoren verbinden Programm mit Datei, Pipes "|" verbinden Programm mit Programm:

```
ls . | grep "some text"
```

- der standard output von ls wird an den standard input von grep weitergeleitet

Man kann mit Pipes beliebig viele Programme aneinanderreihen:

```
ls . | grep "some text" | wc -l
```

Reguläre Ausdrücke

(Regular Expressions, regexes)

Platzhalter: .

Das Symbol . (Punkt) ist ein Platzhalter

für ein einziges, beliebiges Symbol.

Literale: a, b, c, 10, 20...

Ein Literal ist ein regulärer Ausdruck, welcher genau einen konkreten Buchstaben beschreibt. Z.B.: a, b, c, 0, 1, 2...

Jedes Literal und jede Gruppe von Literalen sind reguläre Ausdrücke.

Dies ist also ein regulärer Ausdruck:

abc

Platzhalter: .

Das Symbol . (Punkt) ist ein Platzhalter

für ein einziges, beliebiges Symbol.

Quantifier: *+?{}

- sind spezielle Characters eines regulären Ausdrucks, die angeben, wie oft der Ausdruck, der direkt links von ihnen steht, im regex vorkommt.

Beispiele:

a*	das Literal a, beliebig oft
a+	das Literal a, mindestens ein Mal
B?	das Literal B, ein Mal oder kein Mal
C{2}	das Literal C, genau 2 Mal
C{2,6}	das Literal C, 2 bis 6 Mal
.*	ein beliebiger Buchstabe, beliebig oft

Alternativen: [ ], (|)

Character classes:

[a-z]	ein einziger Kleinbuchstabe von a bis z
[a-zA-C0-9,]	ein einziger Kleinbuchstabe von a bis z oder ein Großbuchstabe von A bis C oder eine Zahl oder ein Komma
[^0-9]	alles außer Zahlen
[-0-9]	ein Minus oder eine Zahl

Alternativen:

(Äpfel|Orangen)	"Äpfel" oder "Orangen"

Anfang / Ende

^ Zeilenanfang

$ Zeilenende

Capturing

- wir fangen diejenigen regex-Teile ab ("capture"), die wir später referenzieren wollen

Zeilenanfang, ein a, irgendein Kleinbuchstabe, nochmal derselbe Kleinbuchstabe:

```
^a([a-z])\1
```

Buchstabe 1, Zahl, Buchstabe 2, dann nochmal Buchstabe 1 und 2:

```
([a-z])[0-9]([a-z])\1\2
```

Search & Replace

Suche nach dem Regex [a-z] in datei.txt:

```
grep -E "[a-z]" datei.txt
```

Suche nach [a-z] in datei.txt und ersetze jedes Vorkommen durch ein a:

```
sed -i "s/[a-z]/a/g" datei.txt
```

Suche und ersetze innerhalb einer Variable:

```
var="abc"
sed "s/[a-z]/a/g" <<< "$var"
```

find & grep

Auf alle mit find gefundenen Dateien grep anwenden:

```
find . -type f -iname "*" -exec grep -EH "some regex" {} \;
```

```
find . -type f -iname "*" -exec sed  ........
```

Quellen

Na ja, Gedächtnis, und diese Seite zu I/O Redirection:

http://www.tldp.org/LDP/abs/html/io-redirection.html

Seriously, check it out, die Seite ist toll!

Ach ja, und dann noch das Buch

"Mastering Regular Expressions" von Jeffrey E. F. Friedl

Danke!