7.Übung

Systemsoftware (SYS)

Christian Baun
cray@unix-ag.uni-kl.de

Hochschule Mannheim – Fakultät für Informatik
Institut für Robotik

16.11.2007

Wiederholung vom letzten Mal

- Zeitgesteuerte Kommandoausführung (`cron`)
- Kommandos zu einer späteren Zeit ausführen (`at`)
- System- und Prozessüberwachung (`top`)
- Regelmäßige Programmausführung (`watch`)
- Systeminformationen ausgeben (`uname`)
- Sortieren (`sort`)
- Umgebungsvariablen anzeigen (`env`, `printenv`)
- Umgebungsvariablen setzen und löschen (`export`, `set`, `unset`)

Heute

- Einführung für Linux/UNIX-Anwender (Teil 6)
  - Textausgaben auf der Shell (`echo`, `printf`, `yes`, `seq`)
  - Inhalt der Shell löschen (`clear`)
  - Mustervergleiche (`sed`)
  - Bearbeiten und Interpretieren von Texten (`awk`)

Kommandos für Textausgaben auf der Shell

- Linux-/UNIX-Betriebssysteme kennen mehrere Kommandos für Textausgaben auf der Shell.
- Diese Kommandos sind besonders bei der Fehlersuche und für das Schreiben von Shell-Skripten hilfreich.
- Einige der wichtigsten Kommandos sind:
  - `echo`
  - `printf`
  - `yes`
  - `seq`
  - `clear`

Einfachen Text ausgeben mit `echo`

`echo` *`[Option] ... [Zeichenkette] ...`*

- Das Kommando `echo` wird verwendet, um einfache Zeichenketten in der Shell auszugeben.

```
$ echo Das ist ein Test
Das ist ein Test
```

- Hilfreiche Optionen von `echo` sind:

`--help`	Eine Hilfe ausgeben.
`-e`	Escape-Zeichen erkennen und interpretieren.
`-E`	Escape-Zeichen ignorieren.
`-n`	Keinen Zeilenvorschub am Ende der Zeile ausgeben.

Escape-Zeichen von `echo`

`\XYZ`	Das ASCII-Zeichen mit dem Oktalwert `XYZ` ausgeben.
`\\`	Backslash.
`\'`	Einfaches Anführungszeichen.
`\"`	Doppeltes Anführungszeichen.
`\a`	Alarm $\Longrightarrow$ erzeugt einen Piepton.
`\b`	Zeichen zurück (Backspace).
`\c`	Zeilenende am Ende der Zeile unterdrücken.
`\f`	Zeilenvorschub.
`\n`	Neue Zeile (Newline).
`\r`	Wagenrücklauf (Carriage return).
`\t`	Horizontaler Tabulator.
`\v`	Vertikaler Tabulator.

Formatierten Text ausgeben mit `printf`

- Das Kommando `printf` bietet eine erweiterte Funktionalität als `echo`.
- Die Syntax ist stark an die C-Funktion `printf()` angelegt.

```
$ printf "Willkommen in der %s-Übung Nr.%d\n" SYS 5
Willkommen in der SYS-Übung Nr.5
```

- Die Formatangaben von `printf` werden in der Manualseite beschrieben.
  $\Longrightarrow$ `man 3 printf`

Formatierten Text ausgeben mit `printf` (2)

- Einige wichtige Formatangaben:

`%d`	Ganze Zahl in Dezimalschreibweise.	`%s`	String.
`%ld`	Long Integer in Dezimalschreibweise.	`%%`	Das Prozentzeichen selbst.
`%o`	Ganze Zahl in Oktalschreibweise.	`%c`	Einzelnes Zeichen.
`%x`	Ganze Zahl in Hexadezimalschreibweise.	`%q`	String mit Shell-Metazeichen.
`%lf`	Gleitkommazahl mit doppelter Genauigkeit.	`%f`	Gleitkommazahl.

- Das Kommando `printf` kann auch mit den Escape-Zeichen von `echo` umgehen.

Einen Text wiederholt ausgeben mit `yes`

- Das Kommando `yes` gibt eine Zeichenkette so lange wiederholt aus, bis es abgebrochen wird.
- Wird `yes` ohne Optionen aufgerufen, gibt das Kommando das Zeichen `y` wiederholt aus.
- Es kann auch ein beliebiger anderer Text ausgegeben werden.

```
$ yes testausgabe
testausgabe
testausgabe
testausgabe
...
```

Nutzen von `yes`

- Das Kommando erscheint auf den ersten Blick nutzlos. Tatsächlich ist `yes` perfekt geeignet, um interaktive Kommandos oder Programme in Shell-Skripten zu verarbeiten.
- Üblicherweise wird die Ausgabe von `yes` in die Eingabe interaktiver Kommandos geleitet.
- Ist das interaktive Kommando beendet, wird auch `yes` beendet.

```
$ yes Nein | interaktives_Kommando
```

Eine Folge von Zahlen ausgeben mit `seq`

`seq` *`[Option] ... Letzter`*
`seq` *`[Option] ... Erster Letzter`*
`seq` *`[Option] ... Erster Plus Letzter`*

- Das Kommando `seq` gibt eine Zahlenfolge aus.
- Genau wie bei `yes` ist das Haupteinsatzgebiet von `seq` die eigene Ausgabe in die Eingabe eines anderes Kommandos zu leiten.
- Ein einfaches Beispiel mit einer oberen Schranke:

```
$ seq 4
1
2
3
4
```

Weitere Beispiele mit `seq` (1)

- Ein Beispiel mit einer unteren und oberen Schranke:

```
$ seq 5 8
5
6
7
8
```

- Ein Beispiel mit einer unteren und oberen Schranke und Schrittweite:

```
$ seq 5 3 20
5
8
11
14
17
20
```

Weitere Beispiele mit `seq` (2)

- Schrittweiten können auch < 0 sein.:

```
$ seq 3 .2 4
3
3,2
3,4
3,6
3,8
```

- Mit der Optionen `-w` erhält die Ausgabe eine einheitliche Breite durch führende Nullen.

```
$ seq -w 8 11
08
09
10
11
```

Weitere Beispiele mit `seq` (3)

- Die Option `-f` *`Formatstring`* ermöglicht es, die Ausgabe speziell zu formatieren.

```
$ seq -f "--== %g ==--" 3
--== 1 ==--
--== 2 ==--
--== 3 ==--
```

- Mit der Optionen `-s` kann eine Zeichenkette als Trennung zwischen den Zahlen festgelegt werden.

```
$ seq -s , 5
1,2,3,4,5
```

Den Inhalt der Shell löschen `clear`

- Das Kommando `clear` löscht den sichtbaren Inhalt des Bildschirms bzw. des Terminals-Fensters.
- `clear` kennt keine Optionen und ignoriert den Versuch Optionen zu übergeben.

Mustervergleiche mit `sed`

- Das Kommando `sed` (**s**tream **ed**itor) ist ein Werkzeug, um Texte auf der Kommandozeile zu verändern.
- Im Gegensatz zu einem interaktiven Texteditor wie `vi`, `emacs` oder `joe` ist `sed` in interaktionslos.
- Die Syntax von `sed` ist eng verwandt mit der Syntax von `vi(m)` und dem Zeileneditor `ed`.
- Die Eingabe wird Zeile für Zeile eingelesen und entsprechend vorgegebener Regeln verändert.
- Häufig wird `sed` eingesetzt, um Texte in Dateien zu ersetzen.
- Die Quelldatei bleibt unverändert. Die Ausgabe wird auf der Shell ausgeben oder in eine Datei geleitet.

Einfache Ersetzung mit sed

```
$ cat test1.txt
Das ist eine alte Datei mit altem Inhalt

$ sed 's/alte/neue/g' test1.txt > test2.txt

$ cat test2.txt
Das ist eine neue Datei mit neuem Inhalt
```

- Die Veränderungsregel 's/alte/neue/g' besagt:
  - In jeder Zeile der Eingabedatei werden die Vorkommen des Regulären Ausdrucks 'alt' durch die Zeichenfolge 'neu' ersetzt.
  - Der sed-Befehl 's' legt fest, dass eine Zeichen-Ersetzung (Substitution) stattfinden soll.
  - Der sed-Befehl 'g' am Ende gibt vor, dass die Veränderung global, also für alle Vorkommen in jeder Zeile, vorgenommen werden soll.

Weitere Beispiele mit `sed` (1)

- Ersetzt mehrfache Leerzeichen durch ein Einziges.
  Das `'\+'` steht für ein- oder mehrmals das vorherige Zeichen:

```
sed 's/ \+/ /g' eingabe.txt > ausgabe.txt
```

- Löscht alle Zeilen, in denen nur Leerzeichen und Tabulatoren vorkommen.

```
sed 's/^[ \t]*$/d' eingabe.txt > ausgabe.txt
```

- Löscht alle Zeilen, in denen der String `Fehler` vorkommt
  Das `'w ausgabe.txt'` weist an, dass die Ausgabe auch in die Datei `ausgabe.txt` geschrieben wird:

```
sed '/Fehler/d w ausgabe.txt' eingabe.txt
```

Weitere Beispiele mit `sed` (2)

- Alle Zeilen auf der Shell ausgeben, die das Wort `Beispiel` enthalten:

```
sed -n '/Beispiel/p' eingabe.txt
```

- Die Zeilen 3 bis 9 der Eingabedatei werden gelöscht:

```
sed '6,9 d' eingabe.txt > ausgabe.txt
```

- Zählt die Zeilen der Eingabedatei (Nachahmung von `wc -l`):

```
sed -n '$=' eingabe.txt
```

- Nummeriert alle Zeilen (linksbündig). Zwischen Zeilennummer und Zeile soll ein Tabulator ausgegeben werden, um den Rand zu erhalten:

```
sed = eingabe.txt | sed 'N;s/\n/\t/'
```

Weitere Beispiele mit `sed` (3)

- Die letzte Zeile der Eingabedatei löschen:

```
sed '$d'
```

- Alle Zeilen der Eingabedatei ausgeben, die kürzer als 5 Zeichen sind:

```
sed -n '/^.\{5\}/!p' eingabe.txt
```

- Alle Leerzeichen und Tabulatoren am Anfang und Ende jeder Zeile löschen:

```
sed 's/^[ \t]*//;s/[ \t]*$//' eingabe.txt
```

- Eine Leerzeile über jeder Zeile einfügen, die `Muster` enthält:

```
sed '/Muster/{x;p;x;}' eingabe.txt
```

Einige `sed`-Befehle

`=`	Die Zeilennummer ausgeben.
`p`	Aktuelle Zeile ausgeben.
`d`	Komplette Zeile löschen.
`i\`	Den (folgenden) Text vor der aktuellen Zeile einfügen.
`a\`	Den (folgenden) Text nach der aktuellen Zeile einfügen
`c\`	Aktuelle Zeile durch den (folgenden) Text ersetzen.
`s`	Suchmuster der aktuellen Zeile ersetzen.
`y`	Buchstaben in der aktuelle Zeile ersetzen.
`r`	Text aus der gegeben Datei anhängen.
`w`	Aktuelle Zeile in die angegebene Datei anhängen.

Umwandlungen mit `awk`

- `awk` ist eine Programmiersprache (Skriptsprache) zur Bearbeitung und Auswertung von einfachen Texten
- Der Name `awk` stammt von den Namen Autoren Alfred V. **A**ho, Peter J. **W**inberger und Brian W. **K**ernighan.
- Grund für die Entwicklung von `awk` war die Handhabung von `sed`, die den Entwicklern nicht zusagte.
- `awk` folgt den Ideen von `sed`, ist aber um viele Eigenschaften reicher. In `awk` finden sich u.a. Kontrollstrukturen, Variablen, Vektoren und Funktionen.
- Wegen der erweiterten Funktionalität wird `awk` nicht mehr als stream editor, sondern als Programmiersprache angesehen.
- Eine erweiterte Version von `awk` ist `gawk`.

Arbeitsweise von `awk` (1)

- Ein `awk`-Programm besteht aus Folgen von Mustern, zugehörigen Aktionen und kann auch Funktionen enthalten:

```
Muster {Aktion}
Muster {Aktion}
...
function funktionsname (Parameter) { Anweisungen }
...
```

- Das Verarbeitunsprinzip von `awk` entspricht dem von `sed`.
- Jede Zeile der Eingabe wird eingelesen und auf das Vorhandensein der Muster geprüft.
- Ist ein Muster vorhanden, wird die zugehörige Aktion ausgeführt.

Arbeitsweise von `awk` (1)

- In jeder Zeile des `awk`-Programms kann entweder das Muster oder die Aktion weggelassen werden.
- Wird das Muster weggelassen, so wird die Aktion für jede Zeile ausgeführt.
- Fehlt die Aktion, wird die jeweilige Zeile (auf die das Muster zutrifft) ausgegeben.
- Anweisungen werden durch neue Zeilen oder durch Semikolon voneinander getrennt.

Interne Variablen von `awk`

- `awk` kennt einige interne Variablen, eine häufig verwendete Auswahl:

`NF`	Anzahl der Felder in der aktuellen Zeile (getrennt durch den Feldtrenner).
`FS`	Feldtrenner (Standardmäßig: Leerzeichen).
`RS`	Zeilentrenner (Standardmäßig: newline).
`FNR`	Aktuelle Zeile der Eingabe.
`NR`	Anzahl der bisher abgearbeiteten Zeilen.
`$n`	Das `n`-te Feld der aktuellen Zeile ($1 \leq n \leq NF$).
`FILENAME`	Name der Eingabedatei.
`ARGC`	Anzahl der Kommandozeilenargumente.
`ARGV`	Array der Kommandozeilenargumente (indexiert von 0 bis `ARGC` - 1).

Ein einfaches awk-Beispiel ohne Aktion

```
$ cat awk_beispiel.txt
10 13
15 12
12 17
19 11

$ awk '$2 > $1' awk_beispiel.txt
10 13
12 17
```

- Es werden alle Zeilen der Datei awk_beispiel.txt ausgegeben, bei denen der Wert im zweiten Feld größer ist als der Wert im ersten Feld.
- Die Felder sind durch Leerzeichen voneinander getrennt.
- Im Beispiel ist keine Aktion angegeben. Es wird automatisch jede Zeile ausgegeben, auf die das Muster ('$2 > $1') zutrifft.

Ein einfaches awk-Beispiel ohne Muster

```
$ cat awk_beispiel.txt
10 13
15 12
12 17
19 11

$ awk '{ print $1 + $2 }' awk_beispiel.txt
23
27
29
30
```

- Hier fehlt das Muster und es ist nur eine Aktion angegeben.
- Die Aktion wird auf alle Zeilen der Datei `awk_beispiel.txt` angewendet und die Ausgabe (Summe der Spalten 1 und 2) ausgegeben.

Ein awk-Beispiel mit Muster und Aktion

```
$ cat awk_beispiel2.txt
10 13 12 18 10
11 14
15 12 10 27
12 17 11
19 18 10 13

$ awk 'NF == 4 { print NR, $NF }' awk_beispiel2.txt
3 27
5 13
```

- Hier sind Muster und Aktion vorhanden.
- Es werden alle Zeilen der Datei verarbeitet, die 4 Felder haben.
- Ausgegeben wird die Zeilennummer (`NR`) und das letzte Feld (`NF`) jeder Zeile, auf die das Muster (`'NF == 4'`) passt.

Nächste Übung:

23.11.2007