Linux und Shell-Programmierung – Teil 4

Prof. Dr. Christian Baun

Fachhochschule Frankfurt am Main
Fachbereich Informatik und Ingenieurwissenschaften
christianbaun@fb2.fh-frankfurt.de

Heute

- Einführung für Linux/UNIX-Anwender (Teil 4)
  - System- und Prozessüberwachung (`top`)
  - Ressourcenauslastung beobachten (`vmstat`)
  - Regelmäßige Programmausführung (`watch`)
  - Systeminformationen ausgeben (`uname`)
  - Sortieren (`sort`)
  - Umgebungsvariablen anzeigen (`env`, `printenv`)
  - Umgebungsvariablen setzen und löschen (`export`, `set`, `unset`)
  - Textausgaben auf der Shell (`echo`, `printf`, `yes`, `seq`, `clear`)
  - Mustervergleiche (`sed`)
  - Bearbeiten und Interpretieren von Texten (`awk`)

Systemressourcen und Prozesse überwachen – top

```
top [Option] ...
```

- Das Kommando top hat eine ähnliche Funktionalität wie ps
- Mit top überwacht man Rechen- und Speicherressourcen sowie Prozesse
- Die Ausgabe von top wird automatisch alle 2 Sekunden aktualisiert
  - Der Zeitraum kann mit der Option -s <Sekunden> festgelegt werden
- Die Ausgabe von top ist in zwei Abschnitte geteilt:
  1. 5 Kopfzeilen mit Informationen zur Uptime, mittleren Last, Anzahl der Prozesse, Speicherverbrauch, CPU- und SWAP-Auslastung, usw.
  2. Tabelle der aktuell laufenden Prozesse mit Informationen zur PID, Benutzer, Priorität, Speicherverbrauch, CPU-Auslastung, Zeit seit Erstellung, Kommandoname, usw.

Einige Tastenkürzel von `top`

- Das Kommando `top` kennt viele Tastenkürzel
- Eine Auswahl:

`Leertaste`	Sofortige Aktualisierung der Ausgabe
`c`	Umschalten zwischen Befehl und Kommandozeile
`i`	Inaktive Prozesse anzeigen oder ausblenden
`l`	Mittlere Last (*load level*) anzeigen oder ausblenden
`m`	Speicheranzeige anzeigen oder ausblenden
`t`	Anzeige der Prozess- und CPU-Zustände anzeigen oder ausblenden
`A`	Sortiert die Prozesse nach Alter (jüngster Job zuerst)
`M`	Sortiert die Prozesse nach Speicherverbrauch
`N`	Sortiert die Prozesse nach Prozess-ID
`P`	Sortiert die Prozesse nach CPU-Belastung
`T`	Sortiert die Prozesse nach gelaufener Zeit
`W`	Schreibt die aktuellen Einstellungen nach `~/.toprc`

Ressourcenauslastung mit `vmstat` beobachten

```
vmstat [Verzögerung]
```

- Das Kommando `vmstat` ist geeignet, die Ressourcenauslastung zu beobachten und so Engpässe zu identifizieren
- `vmstat` sammelt Informationen über:
  - wartende und schlafende Prozesse
  - Kapazität und Nutzung des Arbeitsspeichers
  - Auslagerungsktivitäten des Arbeitsspeichers in Swap-Bereiche
  - Ein-/Ausgehende Daten auf blockorientierte Geräte (Festplatten)
  - Anzahl der Interrupts und Kontextwechsel pro Sekunde
  - Auslastung des Prozessors
- Die Option `Verzögerung` legt die Sekunden fest, in der das Kommando erneut aufgerufen wird

```
$ vmstat
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 0  0 512868  73064  50000 231028    3    3    13    16   64   53 20  4 73  3
```

Ausgabe von `vmstat` im Detail

`r`	Auf CPU-Zeit wartende Prozesse (sollte ≤ 2 sein)
`b`	Nicht wiederbelebare, schlafende Prozesse (sollte nahe 0 sein)
`swpd`	Genutzter, virtueller Arbeitsspeicher
`free`	Freier Arbeitsspeicher
`buff`	Als Festplatten-Puffer genutzter Arbeitsspeicher
`cache`	Als Cache genutzter Arbeitsspeicher
`si`	In den Arbeitsspeicher verlagerter Festplatten-Speicher pro Sekunde
`so`	Vom Arbeitsspeicher auf die Festplatte verlagerter Speicher pro Sekunde
`bi`	Eingehende Daten von blockorientierten Geräten. Blöcke pro Sekunde
`bo`	Ausgehende Daten von blockorientierten Geräten. Blöcke pro Sekunde
`in`	Interrupts pro Sekunde
`cs`	Kontext-Wechsel pro Sekunde
`us`	Zeit-Verbrauch durch User-Prozesse
`sy`	Zeit-Verbrauch durch Kernel-System-Prozesse
`id`	Ungenutzte Prozessor-Zeit
`wa`	Zeit-Verbrauch mit warten auf Ein-/Ausgabe (z.B. Festplatte, Netzwerk)

Regelmäßige Programmausführung mit `watch`

- Häufig beobachtet man Entwicklungen, wie z.B. Anwachsen von Dateien, Systemauslastung, Belegung der Partitionen, usw.
- Mit dem Kommando `watch` können Kommandos in gewünschten Abständen automatisch ausgeführt werden
- Die Voreinstellung ist 2 Sekunden
- Bei jedem Durchgang löscht `watch` zuerst den Bildschirm
- `$ watch df`

```
Every 2,0s: df                                   Thu Oct 25 16:48:08 2007

Dateisystem 1K-Blöcke  Benutzt Verfügbar Ben% Eingehängt auf
/dev/hda3    67406640 58012324   9394316  87% /
tmpfs          647744        0    647744   0% /dev/shm
/dev/hda1     9267572  8100280   1167292  88% /mnt/windows
tmpfs           10240       76     10164   1% /dev
```

Das Kommando `watch` bedienen

- Mit der Option `-n <Sekunden>` kann der Zeitabstand zwischen den Kommandoausführungen von `watch` eingestellt werden
- Um die Unterschiede zwischen den Ausgaben von `watch` hervorzuheben und damit die Lesbarkeit zu erhöhen, existiert die Option `-d`
- Das Tastenkürzel `Strg-C` beendet die Ausführung von `watch`

Systeminformationen ausgeben – uname

- Das Kommando uname kann wichtige Systeminformationen ausgeben:
- Mit der Option -a gibt uname alle Informationen aus

```
$ uname -a
Linux olymp 2.6.18 #3 PREEMPT Tue Jan 2 16:58:43 CET 2007 i686
GNU/Linux
```

- Die Informationen umfassen:
  - Kernelname (Linux)
  - Hostname (olymp)
  - Release-Nummer des Kernels (2.6.18)
  - Kernelversion (#3 PREEMPT Tue Jan 2 16:58:43 CET 2007)
  - Architektur (i686)
  - Name des Betriebssystems (GNU/Linux)

Einzelne Informationen von `uname` ausgeben

- Das Kommando `uname` kennt mehrere Optionen, um die Systeminformationen auch einzeln ausgeben zu können
  - `--help` Eine Hilfe ausgeben.
  - `-a` Alle bekannten Informationen ausgeben
  - `-i` Hardwareplattform ausgeben
  - `-m` Rechnerarchitektur ausgeben
  - `-n` Hostnamen ausgeben
  - `-o` Namen des Betriebssystems ausgeben
  - `-p` Typ des Prozessors ausgeben
  - `-r` Release-Nummer des Kernels ausgeben
  - `-s` Kernelnamen ausgeben.

Sortieren mit `sort`

- Das Kommando `sort` teilt jede Zeile seiner Eingabe in Felder ein
- Als Trenner sind Leerzeichen (Blank) und Tab als Trenner voreingestellt
- Jede Zeile der Eingabe wird über die Felder von links nach rechts sortiert

```
$ cat sortieren
BCDE STUV
ABCD EFGH
WXYZ HIJK
DEFG MNOP
ABCD LMNO
```

```
$ sort sortieren
ABCD EFGH
ABCD LMNO
BCDE STUV
DEFG MNOP
WXYZ HIJK
```

```
$ sort -r sortieren
WXYZ HIJK
DEFG MNOP
BCDE STUV
ABCD LMNO
ABCD EFGH
```

Optionen von `sort`

`--help`	Eine Hilfe ausgeben
`-b`	Führende Leerzeichen und Tabs ignorieren
`-f`	Klein- als Großbuchstaben behandeln
`-n`	Numerisch sortieren (9 kommt vor 10)
`-r`	Ergebnis der Sortierung umkehren
`-u`	Doppelte Zeilen ignorieren
`-o<Dateiname>`	Ergebnis in eine Datei schreiben
`-c`	Nur prüfen, ob die Dateien sortiert sind
`-t<Zeichen>`	`<Zeichen>` als Trenner verwenden
`-k<n>[,<m>]`	Die ersten `<n>` Felder werden ignoriert und die Felder `n` bis `m-1` zum Sortieren verwendet. Die Felder werden mit 1 beginnend nummeriert

Sortierschlüssel angeben mit `sort`

- Mit der Option `+n [-m]` werden die Zeichen zwischen `n` (ausschließlich) und `m` (einschließlich) zum Sortierschlüssel bestimmt
- Wenn `m` fehlt, sind alle Zeichen von `n` bis zum Zeilenende Sortierschlüssel
- Positionen werden in der Form `x.y` angegeben. Das heißt: Zeichen Nummer `y` von Feld Nummer `x`
- Ist kein `y`-Teil angegeben, wird nach dem ersten Zeichen im Feld sortiert
- Beispiele für Sortierschlüssel:

`+2`	Linux ist **ein einfaches Betriebssystem**
`+3.2`	Linux ist e**in** einfaches Betriebssystem
`+2 -3`	Linux ist **ein** einfaches Betriebssystem

Einsatzbeispiele zu `sort`

- Eine Dateiliste erzeugen und anhand der Zeilen in den Dateien sortieren:

```
wc -l * | sort -nr
```

- Die Einträge in der Datei `/etc/passwd` anhand der Benutzer-ID in numerischer Ordnung sortieren:

```
sort -k3 -n -t: passwd
```

Umgebungsvariablen ausgeben – env und printenv

- Umgebungsvariablen sind Variablen der Shell, in denen beliebige Daten gespeichert und wieder ausgelesen werden können
- Welche Umgebungsvariablen aktuell gesetzt sind, erfährt man mit den Kommandos env und printenv

```
user@server:~$ env
TERM=xterm
SHELL=/bin/bash
PS1=\u@\h:\w\$
USER=user
PATH=/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games:
PWD=/home/user/
EDITOR=joe
LANG=de_DE@euro
LANGUAGE=de_DE:de:en_GB:en
DISPLAY=:0.0
...
```

Umgebungsvariablen setzen und löschen

- Umgebungsvariablen anlegen mit `set` oder besser `export` bzw. `setenv`
- Umgebungsvariablen entfernen mit `unset` bzw. `unsetenv`
- Zugriff auf Umgebungsvariablen: Immer mit $ vor dem Namen

```
user@server:~$ export EDITOR=/usr/bin/joe
user@server:~$ printenv | grep EDITOR
EDITOR=/usr/bin/joe
user@server:~$ echo $EDITOR
/usr/bin/joe
user@server:~$ unset EDITOR
user@server:~$ printenv | grep EDITOR
```

- **Problem:** Mit `export` angelegte Umgebungsvariablen sind weg, wenn die Shell beendet wird $\Longrightarrow$ **Lösung:** Eintrag in die Datei `~/.profile`

Beispiel zu Umgebungsvariablen

- Die Umgebungsvariable `PATH` enthält 4 Pfade:

```
user@server:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/bin/X11
```

- Wie kann einfach und schnell der Pfad `/usr/games` hinzugefügt werden?
- Lösung:

```
user@server:~$ export PATH=$PATH:/usr/games
user@server:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
```

- **Wichtig:**

  Umgebungsvariablen anlegen $\Longrightarrow$ Ohne $ vor dem Namen

  Auf Umgebungsvariablen zugreifen $\Longrightarrow$ Mit $ vor dem Namen

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 wichtige Kommandos:
  - `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 Formatangaben:

`%d`	Ganze Zahl in Dezimalschreibweise
`%ld`	Long Integer in Dezimalschreibweise
`%o`	Ganze Zahl in Oktalschreibweise
`%x`	Ganze Zahl in Hexadezimalschreibweise
`%lf`	Gleitkommazahl mit doppelter Genauigkeit
`%s`	String
`%%`	Das Prozentzeichen selbst
`%c`	Einzelnes Zeichen
`%q`	String mit Shell-Metazeichen
`%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 (1/3)

- 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 (2/3)

- 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` (3/3)

- 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 Argumente und ignoriert den Versuch welche zu übergeben
- In das Bash-Shell gibt es noch ein paar Tastenkürzel mit vergleichbarer Funktion:
  - `Strg-l` löscht den Inhalt der Shell genau wie `clear`
  - `Strg-U` löscht den Inhalt der aktuellen Zeile
  - `Strg-L` löscht den Inhalt der Shell außer der aktuellen Zeile
  - `Strg-W` löscht das Wort vor dem Cursor in der aktuellen Zeile
  - `Strg-K` löscht den Inhalt der aktuellen Zeile nach dem Cursor

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 `'alte'` durch die Zeichenfolge `'neue'` 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, die nur Leerzeichen und Tabulatoren enthalten.

```
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
```

- Löscht alle Leerzeichen und Tabulatoren am Anfang und Ende jeder Zeile:

```
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 (1/2)

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

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

- Das Verarbeitungsprinzip 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` (2/2)

- 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