Betriebssysteme

Labor 5

AWK

- ist eine Dienstprogramm, mit der Daten bearbeitet und Berichte erstellt werden
- ist nicht nur ein Textverarbeitungsprogramm, sondern auch eine interpretierte Programmiersprache mit C-ähnlicher Syntax

- sein Name leitet sich von seinen Schöpfern ab: Alfred **A**ho, Peter **W**einberger, Brian **K**ernighan

- erfordert keine Kompilierung und ermöglicht dem Benutzer die Verwendung von Variablen, numerischen Funktionen, Zeichenfolgenfunktionen und logischen Operatoren

AWK

- **Befehlssyntax:**

```
awk [OPTIONS] '/pattern/' [input-file]
awk [OPTIONS] '{action}' [input-file]
awk [OPTIONS] '/pattern/{action}' [input-file]
```

  - **-F** *fs*: um mit `fs` das Standard-Trennzeichen für Eingabefelder zu ändern
  - **-f** *script-file*: um die Befehle aus der Skriptdatei zu erhalten

AWK

- **Wie funktioniert?** `awk` ermöglicht es einem Programmierer, winzige, aber effiziente *Programme in Form von Anweisungen* zu schreiben, die Textmuster definieren (reguläre Ausdrücke), nach denen *in jeder Zeile einer Datei gesucht* werden soll, und die Aktion, die *ausgeführt werden soll*, wenn eine Übereinstimmung in einer Zeile gefunden wird.

- Meistens verwendet für: Erkennung und Verarbeitung von Mustern (Patterns)
  - Es durchsucht eine oder mehrere Dateien, um festzustellen, ob sie Zeilen enthalten, die mit den angegebenen Mustern übereinstimmen, und führt dann die zugehörigen Aktionen aus.

AWK

- `awk` liest und verarbeitet nacheinander alle Zeilen der Eingabedatei
- jede Zeile repräsentiert einen Eingabedatensatz (`input record`)
- **Standard**trennzeichen für Eingabe**sätze**: `CR` (`Carriage Return`)
- der `aktuelle Eingabedatensatz`wird in der internen Variablen `$0` gespeichert
- jeder Eingabedatensatz wird geparst und in Chunks, die `Felder` genannt werden, getrennt
- **Standard**trennzeichen für Eingabe**felder**: `SPACE` oder `TAB`

Eingebaute Variablen (built-in variables)

- **`$0:`** der aktuelle Eingabesatz
- **`$1, $2, ...:`** die Felder des aktuellen Eingabedatensatzes
- **`NR:`** die Gesamtzahl der bisher gesehenen Eingabedatensätze
- **`NF:`** die Anzahl der Felder im aktuellen Eingabedatensatz
- **`RS:`** das Eingabedatensatztrennzeichen
- **`ORS:`** das Ausgabedatensatztrennzeichen
- **`FS:`** das Eingabefeldtrennzeichen

Eingebaute Variablen (built-in variables)

- **`OFS:`** das Ausgabefeldtrennzeichen
- **`OFMT:`** das Format zum Konvertieren von Zahlen in Zeichenfolgen zum Drucken mit print
- **`ARGC:`** die Anzahl der Befehlszeilenargumente
- **`ARGV:`** das Array von Befehlszeilenargumenten
- **`FILENAME:`** der Name der aktuellen Eingabedatei
- **`FNR:`** die aktuelle Datensatznummer in der aktuellen Datei
- **`ENVIRON:`** das Array von Umgebungsvariablen

Vergleichsoperatoren

Operator	Bedeutung	Beispiel
<	weniger als (less than)	`x < y`
<=	kleiner oder gleich als (less than or equal to)	`x <= y`
==	gleich (equal)	`x == y`
!=	ungleich (not equal)	`x != y`
>	größer als (greather than)	`x > y`
>=	größer oder gleich als (greather than or equal to)	`x >= y`
~	entspricht dem regulären Ausdruck (matches the regular expression)	`x ~ /regexp/`
!~	entspricht nicht dem regulären Ausdruck (does not match the regular expression)	`x !~ /regexp/`

Operatoren

- logische Operatoren:

```
&&   ||   !
```

- arithmetische Operatoren:

```
+   -   *   /   %   ^
```

- Zuweisungsoperatoren:

```
=   +=   -=   *=   /=   %=   ^=
```

- bedingte Ausdrücke:

```
condition ? expresion1 : expresion2
```

-> ist äquivalent mit:

```
if (condition)
    expresion1
else
    expresion2
```

Eingebaute Funktionen

- **`length(sir)`**
  - bezeichnet die Länge der Zeichenfolge sir
  - `length <=> length($0)`
- **`substr(s,p,n)`**
  - bezeichnet der Teilzeichenfolge von `s`, der an Stelle `p` beginnt und die Länge `n` hat
- **`index(s1,s2)`**
  - gibt die Stelle zurück, an der `s2` in `s1` erscheint, oder `0` sonst

Eingebaute Funktionen

- **`sprintf(format, arg1,..)`**
  - gibt die Zeichenfolge zurück, die printf als Ergebnis in C drücken würde
- **`split(s,a,c)`**
  - `s` ist eine Zeichenfolge, `a` ein Array und `c` ein Zeichen.
  - eilt die Zeichenfolge `s` in Felder ein und betrachtet als Trennzeichen das Zeichen `c`. Wenn `c` fehlt, nimmt man das Standardtrennzeichen `FS` an.
  - die resultierenden Werte sind als Werte in Tabelle `a` angegeben.

Scripts

- **`BEGIN`**: Befehle werden nur einmal ausgeführt, **`BEVOR`** der erste Eingabedatensatz gelesen wird
- **`END`**: Befehle werden nur einmal ausgeführt, **`NACHDEM`** alle Eingaben gelesen wurden
- **`{}`** zwischen **`BEGIN`** și **`END`**: Befehle werden **`für jeden`** Eingabedatensatz ausgeführt

Beispiele

Betrachten Sie die folgende Textdatei:

noten.txt

```
Ana ASC 80
Mihai Algebra 90
Ana FP 89
Andrei MAP 87
Ion Algebra 77
Maria FP 95
Andrei Geometrie 85
```

Beispiele

- Standardverhalten von Awk: drückt jede Zeile aus der angegebenen Datei

```
awk '{print}' noten.txt
```

- Drückt die Zeilen, die mit dem angegebenen Muster übereinstimmen

```
awk '/FP/' noten.txt
awk '/FP/{print}' noten.txt
awk '/FP/{print $0}' noten.txt
```

Bemerkung:

- Für jeden Datensatz, das heißt Zeile, teilt der Befehl `awk` den durch Leerzeichen begrenzten Datensatz standardmäßig auf und speichert ihn in den Variablen `$n`.
  - **Zum Beispiel**: Wenn die Zeile 3 Wörter enthält (wie in unserem Beispiel), wird sie in `$1`, `$2` bzw. `$3` gespeichert. Außerdem steht `$0` für die gesamte Zeile.

Beispiele

- Aufteilen einer Zeile in Felder:

```
awk '{print $1, $3}' noten.txt
```

- Änderung des Standardtrennzeichens für Eingabefelder:

```
awk -F: '{print $1}' users.txt
awk -F: '{print NR, $1}' users.txt
```

  - wenn man mehrere Arten von Trennzeichen haben:
    - `awk -F'[:/]' '{print $2}' noten2.txt`

- Anzeigen der Zeilennummer (Verwendung von NR-integrierten Variablen):

```
awk '{print NR,$0}' noten.txt
```

- Anzeigen des letzten Felds (Verwendung der in NF-integrierten Variablen)

```
awk '{print $NF}' noten.txt
```

Beispiele

- Anzeigen der Zeilen von 2 bis 5 (Verwendung von NR-integrierten Variablen)

```
awk 'NR==2, NR==5 {print NR,$0}' noten.txt
awk 'NR>=2 && NR<=5 {print NR,$0}' noten.txt
```

- Bestimmen der maximalen Länge der Linie in der Datei:

```
awk '{ if (length($0) > max) max = length($0) } END { print max, $0 }' noten.txt
```

- Suchen/Überprüfung einer bestimmten Zeichenfolge in einer bestimmten Spalte:

```
awk '{ if($2 == "FP") print $0;}' noten.txt
```

- Zählen der Zeilen in einer Datei:

```
awk 'END { print NR }' noten.txt
```

Aufgaben

1. Verwenden Sie den Befehl `awk`, um den Benutzername, Benutzer-ID und dem Ausgangsverzeichnis aus der Datei `users.txt` anzuzeigen.

HINWEIS: das Format eines Datensatzes aus der users.txt Datei ist der Form:

```
Name:Kennwort:Benutzer-ID:Group-ID:Vollname:Ausgangsverzeichnis:...
```

2. Sei die folgende Textdatei: **`datei.txt`**.

```
Hier ist Zeile 1.
Hier ist Zeile 2.
Hier ist Zeile 3.
Hier ist Zeile 4.
Hier ist Zeile 5.
```

Führen Sie den folgenden Befehl aus:

```
awk 'BEGIN{FS=","}{print $1,"FNR="FNR,"NR="NR}END{print "Insgesamt",NR,"verarbeitete Zeilen"}' datei.txt datei.txt datei.txt
```

Was fällt Ihnen auf?

Aufgaben

3. Verwenden Sie den Befehl `awk`, um die erste und die zweite Spalte aus den ersten 10 Zeilen von `users.txt` anzuzeigen.

4. Sei die folgende Textdatei **`summe.txt`**.

```
1
4
58
5
9
11
```

Verwenden Sie den Befehl `awk`, um die Summe aller Zahlen aus der `summe.txt` Datei zu berechnen.

Aufgaben

5. Drucken Sie das Ausgangsverzeichnis der Benutzer aus, deren Benutzernamen mit einem Vokal beginnen. (Verwendung der `users.txt` Datei)

6. Drucken Sie den vollständigen Namen der Benutzer mit geraden Benutzer-IDs. (Verwendung der `users.txt` Datei)

7. Zeigen Sie die Benutzernamen aller Benutzer an, deren vollständiger Name länger als 20 Zeichen ist. (Verwendung der `users.txt` Datei)

8. Drucken Sie alle Benutzer in einer geraden Zeile mit einer Gruppen-ID von weniger als 20 aus, geben Sie den `awk` - Befehl in einer Datei an. (Verwendung der `users.txt` Datei)
   HINWEIS: `Der Befehl zum Ausführen der Datei sieht folgendermaßen aus:`
   `awk -F: -f awkDatei users.txt`

Aufgaben

9. Zeigen Sie die Summe aller Benutzer-IDs an, geben Sie den `awk` - Befehl in einer Datei an. (Verwendung der `users.txt` Datei)

10. Zeigen Sie das Produkt der Differenz zwischen der Benutzer-ID und der Gruppen-ID an, geben Sie den `awk` - Befehl in einer Datei an. (Verwendung der `users.txt` Datei)

11. Verwenden Sie den Befehl `awk`, um die Quadrate der Zahlen 1,2,3,4,5 anzuzeigen.

12. Sei die folgende Textdatei: **`phone.txt`**.

```
0746:133:225
0746:132:225
0256:141:133
```

Verwenden Sie den Befehl `awk`, um das Trennzeichen "`:`" in das Trennzeichen "`-`" zu ändern.

Ressourcen

- Anweisungen:

  `http://www.grymoire.com/Unix/AwkRef.html`
- Eingebaute Funktionen:

  `http://www.grymoire.com/Unix/AwkRef.html`
- awk manual:

  `https://linux.die.net/man/1/awk`
- awk tutorial:

  `http://www.grymoire.com/Unix/Awk.html`