# Sistemi Operativi

AAF - Secondo anno - 3CFU

A.A. 2024-2025

Corso di Laurea in Matematica

# AWK. Costrutti IF e FOR.

Annalisa Massini

Dipartimento di Informatica
Sapienza Università di Roma

# Argomenti trattati

1. Comandi su file di testo
   - Il comando `awk`
2. Costrutti
   - Il costrutto IF
   - Il costrutto FOR

## Altri comandi

# Il comando awk

# Comando awk

- L'utility awk serve per processare file di testo secondo un programma specificato dall'utente
- awk deriva dalle iniziali dei cognomi dei suoi autori, Alfred **A**ho, Peter **W**einberger e Brian **K**ernighan
- awk legge i file riga per riga ed esegue una o più **azioni** su tutte le linee che soddisfano certe **condizioni**
- Azioni e condizioni sono descritte tramite una sequenza che definisce un programma
- Nei moderni Linux, awk è un link a versioni più recenti come gawk e mawk

# Comando awk

- La sintassi di awk è del tipo:
  `awk <opzioni...> 'codice awk' <file-di-testo>`
  - `codice awk` è un programma che specifica azioni e condizioni
  - oltre a comparire sulla linea di comando tra singoli apici `codice awk` può essere in un file (awk script) specificato con l'opzione `-f`
  - Ogni linea del file in input `file-di-testo` è vista come una sequenza di campi separati da tab e/o spazi
  - L'opzione `-F` serve per specificare un carattere separatore sostitutivo
- Più estesamente si ha: `awk [-F separatore] [-f file_awk] [-v var=var...] [programma_awk] [file...]`

# Comando awk

- Un programma awk si può presentare come una lista del tipo:
  `[condizione1 [, condizione2, ...]] {azioni}`
- Per ogni riga del file in input, vengono valutate le condizioni e, se il risultato è vero, vengono eseguite i comandi del programma
- Per ogni riga possono essere eseguiti 0, 1 o più programmi
- Se ci sono più condizioni, separate da virgola, allora il programma viene applicato a tutte le righe che si trovano tra la prima riga che soddisfa la prima condizione e l'ultima riga che soddisfa l'ultima condizione

# Comando awk

- Non mettere la condizione equivale a dire che il rispettivo programma va eseguito per tutte le righe
- Prima di essere eseguire condizioni e programmi, ogni riga del testo viene spezzata in campi (cioè ridotta in **token** o *tokenizzata*), usando un *field separator* `FS`
  - di default, `FS` è un qualunque spazio, ma con l'opzione `-F` lo si può ridefinire ad un qualunque carattere
  - lo stesso effetto lo si può ottenere assegnando un valore ad `FS` all'interno di un programma

# Comando awk

- All'interno di condizioni e programmi si possono usare paramentri speciali, come ad esempio (ma non solo):

  `FNR` : numero di riga del file attuale
  `NR` : numero di riga tra tutti i file
  `ARGIND` : indice del file attuale (il primo ha indice 1)
  `NF` : numero di campi
  `FS` : separatore di campi
  `$1, $2, ...` : parametri posizionali per i campi della linea corrente (il numero di campo è tra 1 ed `NF`)
  `$0` : l'intera riga non spezzata

- si possono inoltre usare tutte le variabili eventualmente specificate con l'opzione `-v`

# Comando awk

- Le **condizioni**, possono essere definite come segue:
  - espressioni contenenti operatori logici e/o relazionali
  - `(var)_o_(const) cmp (var)_o_(const)`: dove `cmp` è un operatore di confronto (`==`, `!=`, `>`, `<`, `>=`, `<=`)
  - le precedenti condizioni sono atomiche; possono essere combinate con AND (`&&`), OR (`||`) e NOT (`!`), e raggruppate con le normali parentesi
  - ci sono 2 condizioni speciali: `BEGIN` (vale solo prima della prima riga del primo file) e `END` (vale solo dopo l'ultima riga dell'ultimo file)

# Comando awk

- Le **azioni** che costituiscono i programmi possono essere definite seguendo la sintassi del C/Java
  - assegnamenti con =
  - test di uguaglianza con ==
  - `for (init; cond; iter) istruzioni`
  - `while (cond) istruzioni`
  - `do istruzioni while (cond)`
  - `break` e `continue`
  - `if (cond) istruzioni`
  - `if (cond) istruzioni; else istruzioni`
  - se `istruzioni` è un blocco che contiene più istruzioni, va racchiuso dalle parentesi graffe
  - `exit` salta alla riga di comando successiva

# Comando awk

## Esempi per cominciare

- Creare un file amici.txt con linee composte da `nome, numero` (età, numero di telefono, data, ecc.)
  - Il comando `awk '/Anna/ { print $2 }' amici.txt`
    - `awk { print $n }` stampa il campo n di ogni linea di un file
    - `/nome/` seleziona tutte le linee che contengono nome
  - Scrivere `awk '($2>x) { print $2, $1 }' amici.txt`
    - Cosa viene visualizzato?
  - Scrivere `awk '($2>x && $2<y) { print NR, $2, $1 }' amici.txt`
    - Cosa viene visualizzato?

# Comando awk

## Esempi per cominciare

- Programma awk per sommare i valori del quinto campo di ogni record
  - `awk ' { totale += $5 } END { print "totale " totale " byte" }' file.txt`
- Programma awk che eseguito su `file.txt` estrae le righe pari oppure dispari
  - `awk '(NR % 2) { print }' file.txt`
  - `awk '((NR % 2) - 1) { print }' file.txt`
- Programma awk che eseguito su `file.txt` modifica il delimitatore nel contenuto del file (ad esempio nelle date del tipo gg/mm/aaaa produce gg-mm-aaaa)
  - `awk '$1=$1' FS="/" OFS="-" file.txt`

# Comando awk

## Esempi per cominciare

- Programma (*script*) awk che eseguito su `file.txt` stampa il contenuto del file tutto su di una riga:
  - `awk 'BEGIN {RS="\n"; ORS=" ";print "\n"} {print $0} END {print "\n \n"}' file.txt`
- Programma awk che eseguito su `file.txt` stampa il contenuto del file fino alla riga contenente la stringa **parola**
  - `awk '{print} /parola/ {exit}' file.txt`

# Comando awk

- Alcune funzioni utilizzabili:
  - `length(s)`: ritorna la lunghezza della stringa `s` (il numero di elementi, se `s` è un array, ma non in tutte le versioni di `awk`)
  - `split(s, a, sep)`: tokenizza la stringa `s` nell'array `a` (distruggendolo se già esisteva; il primo indice è 1), usando il separatore `sep` (può non essere dato, e allora si usa `FS`); ritorna il numero di token ottenuti
  - `tolower(s)` e `toupper(s)`: ritornano la stringa `s` con le lettere tutte minuscole o tutte maiuscole
  - `int(d)`, `log(d)`, `exp(d)`, `sqrt(d)`, funzioni standard (la `int` non arrotonda, ma tronca)

# Comando awk

## Esempi per cominciare

- Programma awk che eseguito su file.txt stampa tutte le righe del file che superano la lunghezza di 40 caratteri
  - `awk 'length($0) > 40 { print $0 }' file.txt`
- Programma awk che eseguito su un file contenente nomi e date in formato gg/mm/aa consente di estrarre l'anno e il nome:
  - `awk '{ split($0,x,"/"); print x[3], $1 }' file-nomi`

## Esempi di costrutti

# Il costrutto IF

# Struttura di controllo condizionale if

- Una struttura di controllo condizionale consente di valutare un'espressione logica e, sulla base del suo valore (vero o falso), eseguire determinate istruzioni
- La forma più semplice di una struttura di controllo è:

  **se** condizione **allora**
  istruzioni
  **fine-condizione**

- Nel linguaggio Bash questa struttura viene implementata dalle istruzioni `if` ... `then` ... `fi` secondo la seguente sintassi:

  **if** lista comandi **then**
  lista comandi
  **fi**

# Struttura di controllo condizionale if

- Esempio di **if**

```
if [[ -e originale.txt ]] ; then
  cp originale.txt copia.txt
  echo "Copia riuscita."
fi
```

  - la condizione -e risulta vera se il file che la segue esiste

- Esempio di **if-then-else**

```
if [[ -e originale.txt ]] ; then
  cp originale.txt copia.txt
  echo "Copia riuscita."
else
  echo "Il file non esiste."
fi
```

  - notare che gli if si possono annidare oppure si può usare `eli`

## Esempi di costrutti

# Il costrutto FOR

# Struttura di controllo iterative

- Per eseguire più operazioni con uno script di poche righe, si usano strutture di controllo iterative
- I costrutti iterativi permettono di ripetere un certo blocco di comandi più volte, fino a quando non risulta verificata l'espressione logica usata come istruzione di controllo
- Bash mette a disposizione tre istruzioni principali per la realizzazione di iterazioni (cicli): le istruzioni **for**, **while** e **until**
- Per tutte e tre, all'inizio del blocco di istruzioni da ripetere, viene valutata una condizione logica

# Struttura iterativa for

- L'istruzione `for` consente di implementare una struttura di controllo usando una variabile di controllo in una lista di valori o di un intervallo numerico e ha la seguente struttura:

  **for** variabile **in** lista-di-valori
  **do**
  istruzioni
  **done**

- L'istruzione `for` si può anche usando contatore e in tal caso ha una struttura simile al C in cui si hanno tre parametri tra doppie parentesi tonde separati da un punto e virgola del tipo:

  **for** (( espr1 ; espr2 ; espr3 ))
  **do**
  istruzioni
  **done**

# Esempi

- Esempio di **for**

```
for i in 1 2 3 4 5
do
  echo "Numero $i"
done
```

  - La variabile *i* assume ciascuno dei valori indicati dopo la parola chiave `in` nello stesso ordine
  - Per scorrere una lista di numeri interi è possibile anche usare la scorciatoia 1..5

- Esempio di **for**

```
for i in 1 2 topolino pippo 3 pluto
do
  echo "i ha valore $i"
done
```

  - i valori assunti dalla variabile i possono essere di qualsiasi tipo

# Esempi

- Esempio di **for**

```
for (( i = 0 ; i <= 20 ; i += 2 )) ; do
  echo "i ha valore $i"
done
```

  - per definire il ciclo `for` si possono usare le espressioni aritmetiche, separate da ;

- Esempio di **for**

```
for (( i=1 , j=1 ; i<=34 ; i+=j, j+=i )) ; do
  echo -n "$i" "$j "
done
echo
```

  - è possibile definire cicli su sequenze di numeri arbitrarie: nell'esempio la sequenza di valori stampata corrisponde ai primi 10 elementi della successione di Fibonacci