Laboratorio di sistemi operativi
A.A. 2010/2011
Gruppo 2
Gennaro Oliva
10
Awk scripting

Argomenti e variabili

- In awk è possibile assegnare un valore ad una variabile all'atto dell'esecuzione del programma sulla linea di comando
- La sintassi è la stessa che si usa nei programmi:

  variabile=valore
- L'assegnazione si può effettuare all'inizio del programma con l'opzione -v o tra i file di input
- Esempio:

```
$ awk -v sconto=10 '{print $2,\
($3*cambio)*((100-sconto)/100),valuta}' \
valuta=”€” cambio=1 listino.txt \
valuta="£" cambio=0.882098 listino.txt \
valuta="$" cambio=1.4426 listino.txt
```

Modificare la variabile FS

- La variabile FS può essere modificata anche utilizzando l'opzione -F che accetta come argomento un'espressione regolare e la memorizza in FS
- Esempio

```
$ awk -F '[:,]' '{print $5}' \
/etc/passwd
```

e quivale a

```
$ awk 'BEGIN {FS="[:,]"} ; \
{print $5}' /etc/passwd
```

Redirezione

- I comandi print e printf stampano di default su standard output, ma possono essere rediretti verso qualsiasi file con la sintassi che si usa per la shell
- L'istruzione:

  ```
  print elementi > output-file
  ```

  stampa l'ouptut del comando print nel file specificato
- Se il file non esiste viene creato, in caso contrario il contenuto del file viene cancellato alla prima scrittura, mentre le successive scritture, nell'ambito dello stesso programma, avvengono in modalità append

Uso della redirezione

- Il comando che segue crea un file con gli username degli utenti che hanno uid maggiore o uguale a 1000

```
$ awk -F : '$3 >= 1000 \
{print $1 > "usernames.txt"} \
/etc/passwd
```

- Il file usernames.txt se esistente viene cancellato soltanto alla prima scrittura
- Si noti che a differenza di quanto succede con la shell, nonostante ad ogni record corrispondente venga eseguita l'istruzione

```
print $1 > "usernames.txt"
```

il file non viene sovrascritto

Redirezione

- Il nome del file output-file può essere una qualsiasi espressione: il valore dell'espressione viene convertito in stringa ed usato come nome per il file
- Se si vuole scrivere in un file chiamato output-file si dovrà pertanto utilizzare la sintassi

```
print elementi > "output-file"
```

Redirezione

- L'istruzione:

  ```
  print elementi >> output-file
  ```

- stampa l'ouptut del comando print nel file specificato in modalità append
- L'istruzione

  ```
  print elementi | comando
  ```

  redirige l'output nello standard input del comando specificato come espressione
- Anche in questo caso si utilizzano i doppi apici se si desidera indicare direttamente il comando:

  ```
  print $1 | "sort"
  ```

Espressione come nome file

- L'esempio che segue mostra come utilizzare un'espressione come nome del file da creare

```
$ awk \
'{print $2,($3*cambio),valuta > \
"listino-" paese ".txt" }' \
valuta="€" cambio=1 paese="it" \
listino.txt \
valuta="£" cambio=0.882098 paese="uk" \
listino.txt \
valuta="$" cambio=1.4426 paese="us" \
listino.txt
```

if-then-else

- La sintassi del costrutto if-then-else in awk è la seguente:

```
if (condizione)
  istruzione
else
  istruzioni-alternative
```

- La condizione else è opzionale
- Se l'istruzione da eseguire è singola può anche essere messa subito dopo la condizione:

```
if (condizione) istruzione
```

- Se si vogliono eseguire più istruzioni all'interno del corpo dell'if è necessario racchiuderle in parentesi graffe

```
if (condizione){
  istruzioni
  ...
}
```

Esempio d'uso di if

- Supponiamo di voler processare il file delle presenze, così strutturato:

```
presenze:matricola:nome:cognome:email
```

- `$ awk -F : '{if ($1 >= 1) print $3,$4,$5}' presenze`
- Se si vogliono eseguire più istruzioni all'interno del corpo dell'if:

```
#!/usr/bin/awk -f
if ($1 >= 2) {
   print $2,$3,$4,"Ammesso"
   print "Complimenti!" | "mail -s \"Matricola " \
   $2 " ammessa alla prova intercorso\" " $5
   print $2 > "matr-ammesse"
}
```

while

- La sintassi del costrutto while in awk è la seguente:

```
while (condizione)
  istruzione
```

- Se l'istruzione da eseguire è singola può anche essere messa subito dopo la condizione:

```
while (condizione) istruzione
```

- Se si vogliono eseguire più istruzioni all'interno del corpo del while è necessario racchiuderle in parentesi graffe

```
while (condizione){
  istruzioni
  ...
}
```

Esempio d'uso di while

```
#!/usr/bin/awk -f
# stampa i field in colonna
{
  i=1
  while ( i <= NF ) {
    print $i
    i++
  }
}
```

do-while

- L'istruzione while può essere utilizzata anche nel costrutto do-while in cui il test della condizione viene effettuato alla fine del ciclo e pertanto il corpo delle istruzioni viene sempre eseguito almeno una volta

```
do
   istruzioni
while (condizione)
```

for

- La sintassi del costrutto for in awk è la seguente:

```
for (inizializzazione; condizione; incremento)
   istruzioni
```

- L’interprete awk, esegue una sola volta l’inizializzazione e poi esegue il corpo del for fino a quando la condizione è vera, effettuando l'incremento al termine di ogni passo
- La condizione è una qualsiasi espressione booleana non necessariamente dipendente dall'indice
- L’incremento può modificare l’indice in modo arbitrario e può agire su una sola variabile
- Non è possibile inizializzare più variabili nell'inizializzazione (se non scrivendo i=j=valore)

Esempio d'uso del for

- Esempio d'uso del for per stampare i field in colonna
- `$ awk '{for (i=1;i<=NF;i++) print $i}'`

break e continue

- L'istruzione break interrompe l’esecuzione di un ciclo iterativo for, while o do-while proseguendo con la prima istruzione al di fuori al ciclo
- L'istruzione continue interrompe l’esecuzione di un'iterazione di un ciclo for, while o do-while e salta al passo successivo

Esempio di uso di break

```
#!/usr/bin/awk -f
# trova la prima occorrenza di word
# su ogni linea del file
{ i=0
  while (i <= NF) {
    if ( word == $i ) {
      print word, "linea", NR
      break
    }
    i++
  }
}

$ ./trova word=ciao
```

next

- L'istruzione next interrompe l’elaborazione del record corrente e comunica all'interprete di leggere e processare il record successivo
- Si noti che next comunica all’interprete di saltare non solo le istruzioni ma anche le eventuali regole successive presenti

```
...
# La regola che ignora tutte
# le linee che cominciano con '#'
/^#/ { next }

...
```

exit

- L'istruzione exit termina l’esecuzione di un programma awk
- Quando l'istruzione viene invocata nel blocco BEGIN o in una regola del programma, l'interprete esegue comunque il blocco END
- exit accetta un parametro intero opzionale che consente di specificare l'exit value dell’interprete
- exit invocato senza parametri, specifica l'exit value `0`

Esempio d'uso di exit

```
#!/usr/bin/awk -f
# Trova la prima occorrenza di word nel
# file e poi esce
{
  for ( i=1 ; i<=NF ; i++) {
    if ( $i == word ) {
      FOUND=1
      exit 0
    }
  }
}
END {
if (FOUND) print word,"trovata su",NR
}
```

```
$ trovaprima word=ciao
```

Array associativi

- Un array associativo è un struttura dati che associa ad ogni valore un chiave unica che lo identifica univocamente
- A differenza di quanto avviene per gli array classici le chiavi non sono necessariamente numeri interi consecutivi
- Diversi linguaggi di programmazione supportano gli array associativi in modo diretto (awk, perl, python, ...) o con librerie (C++, Java, ...)
- In pratica ogni elemento di un array associativo è una coppia (chiave, valore)

Array associativi in awk

- In awk non è necessario dichiarare la lunghezza di un array
- È possibile aggiungere elementi all'array in qualsiasi momento, indipendentemente dal valore della chiave
- All'interno dello stesso array si possono memorizzare stringhe e valori numerici interi o in virgola mobile
- Le chiavi di uno stesso array possono essere stringhe o valori numerici (interi o in virgola mobile), ma vengono sempre memorizzate come stringhe

Array associativi in awk

- I nomi per gli array seguono le stesse regole di quelli delle variabili scalari: composti esclusivamente dai lettere cifre e '_' e non puoò iniziare con una cifra.
- Per referenziare l'elemento dell'array associato ad una determinata chiave si utilizza la sintassi array[chiave] simile a quella degli array classici
- Nell'assegnazione ad esempio si può scrivere

  ```
  array[chiave]=valore
  ```

- Le chiavi sono case-sensitive quindi array[“x”] e array[“X”] sono elementi diversi dell’array

Array associativi

- Le assegnazioni generano automaticamente gli elementi
- Il riferimento ad una chiave non presente restituisce la stringa vuota “”
- La cancellazione di un elemento o di un array si effettua mediante l'istruzione delete

  delete array[chiave]
- Se non si specifica la chiave si cancella tutto l'array

Uso di array associativi

- Nel file delle presenze degli studenti i record sono così strutturati:
  `presenze:matr:nome:cognome:email`
- Per memorizzare i dati in tre array associativi utilizzando la matricola come chiave

```
…
{
   nome[$2]=$3
   presenze[$2]=$1
   cognome[$2]=$4
   email[$2]=$5
}
…
```

Array associativi e cicli for

- Per referenziare tutti gli elementi e le chiavi dell'array all'interno di un ciclo for è possibile utilizzare la seguenti sintassi:

```
for (var in array)

istruzioni
```

- All'interno del ciclo la variabile var assume ad ogni passo il valore di una delle chiavi presenti nell'array e per accedere ai valori corrispondenti si usa l'espressione array[var]
- L'ordine con cui vengono elencate le chiavi dipende dall'implementazione e non è predicibile

Uso del for con array associativi

Nel file delle presenze degli studenti i record sono così strutturati:
`presenze:matr:nome:cognome:email`

- Per stampare i nominativi di coloro a cui non è associata alcuna email poiché la registrazione non è andata a buon fine

```
{
  for ( i in email )
    if (email[i] == "")
    print nome[i],cognome[i]
}
```

Verifica l'esistenza di una chiave

- L'espressione

  (chiave in array)

  consente di verificare all'interno dell'array c'e' un elemento corrispondente alla chiave
- Restituisce `0` (falso) se la chiave non e' presente, e `1` (vero) in caso contrario

Uso della verifica di esistenza

- Per verificare l'esistenza di una matricola nell'array in fase di memorizzazione si può utilizzare:

```
{
  if (!($2 in nome)) {
    nome[$2]=$3
    presenze[$2]=$1
    cognome[$2]=$4
    email[$2]=$5
  }
  else
    print “matricola”,$2,”presente”
}
```

Funzioni built-in

- Il linguaggio awk forisce una serie di funzioni built-in di varia natura
- Per invocarne l'esecuzione è sufficiente scriverne il nome seguito dalle parentesi tonde e gli argomenti
- Esempio:

  y=sin(x)

- Nelle funzioni che accettano più di un argomento all'atto dell'invocazione questi vengono separati dal carattere virgola
- Alcune funzioni accettano un numero variabile di argomenti, poiché alcuni di essi sono opzionali

Funzioni numeriche

- int(x) restituisce l'intero più prossimo a x compreso tra 0 e x (int(3.9)=3 e int(-3,9)=-3
- sqrt(x) restituisce la radice quadrata ($x \geq 0$)
- exp(x) restituisce e^x
- log(x) restituisce il logaritmo naturale ln(x)
- rand() restituisce un numero random compreso tra 0 e 1 estremi esclusi
- srand(x) imposta il seme della sequenza random, semi uguali generano sequenze uguali

Manipolazione delle stringhe

- index(string, sub) restituisce la posizione della sotto-stringa sub all'interno della stringa string.
- index(“gennaro.oliva@cnr.it”,”@”) restituisce 14 ovvero la posizione del carattere @ all'interno dell'indirizzo email
- Se la sotto-stringa non viene localizzata, la funzione restituisce `0`
- length([stringa]) restituisce il numero di caratteri contenuti nella stringa, e qualora la stringa sia un numero restituisce il numero di caratteri necessari a rappresentare la stringa
- length("gennaro.oliva@cnr.it") è 20 mentre lenght(7*22) è 3 ovvero il numero di cifre del risultato 154

match

- match(stringa, regexp) cerca all'interno di stringa la sotto-stringa più lunga e più a sinistra corrispondente all'espressione regolare specificata
- Restituisce l'indice del carattere iniziale della sotto-stringa corrispondente che viene anche memorizzato nella variabile built-in RSTART
- La lunghezza dell'espressione regolare corrispondente viene memorizzata nella variabile built-in RLENGHT
- Nel caso in cui non ci sia corrispondenza match restituisce `0` e RLENGHT memorizza -1

substr e split

- substr(orig, inizio [, lung]) restituisce la sotto-stringa di orig che parte dal carattere con indice inizio e ha lunghezza lung
- split(stringa, array [, sep]) divide una stringa in parti separate dall'espressione regolare sep (o da FS se questa è omessa) e memorizza le varie parti nell'array
- La prima parte della stringa viene memorizzata in array[1], la seconda in array[2], ...

Uso di match e substr

- La regola che segue accetta in input due field per record, una stringa ed un'espressione regolare e stampa la parte di stringa corrispondente all'espressione regolare specificata

```
{
  if(match($1,$2))
    print substr($1,RSTART,RLENGTH)
}
```

sub e gsub

- sub(regexp, rimpiazzo [, target]) modifica il valore di target (o di `$0` se questi è omesso sostituendo la sotto-stringa più lunga e più a sinistra corrispondente all'espressione regolare specificata con rimpiazzo
- Restituisce `1` oppure `0` a seconda che sia stata effettuata la sostituzione oppure no
- gsub(regexp, replacement [, target]) opera come sub, ma globalmente modificando tutte le sotto-stringhe più lunghe e più a sinistra ad intersezione vuota corrispondenti all'espressione regolare specificata che riesce a trovare
- Restituisce il numero di sostituzioni effettuate

sub e gsub con il carattere '&'

- Come accade per sed se il carattere speciale & appare nel rimpiazzo, questi viene sostituito con la sotto-stringa corrispondente all'espressione regolare trovata
- Per inserire il carattere '&' nel rimpiazzo senza che questi venga interpretato è necessario farlo precedere dal carattere '\' e lo stesso vale per il carattere '\'
  `\\ → \`
  `\& → &`
- sub e gsub operano su un buffer esistente quindi il terzo parametro, quando viene passato, deve essere una variabile (o l'elemento di un array) altrimenti la sostituzione non può avvenire

tolower e toupper

- tolower(stringa) restituisce una copia della stringa con le lettere maiuscole convertite in minuscolo
- toupper(stringa) restituisce una copia della stringa con le lettere minuscole convertite in maiuscolo
- Per rendere minuscoli i nomi e i cognomi degli studenti registrati

```
$ awk -F : '{print $1 ":" $2 ":" \
  tolower($3) ":" \
  tolower($4) ":" $5}'
```