Le funzioni di shell

- La bash supporta la programmazione procedurale e prevede la possibilità di definire funzioni utilizzando le sintassi alternative:

  function nome {

  lista-comandi

  }

  oppure

  nome ( ) {

  lista-comandi

  }

Esecuzione di una funzione di shell

- Per eseguire una funzione è sufficiente invocare il suo nome dalla shell o all'interno di uno script come se fosse un normale comando
- Se si definisco più funzioni con lo stesso nome viene eseguita l'ultima funzione definita pertanto è opportuno che le funzioni di shell abbiamo nomi distinti tra loro
- Le funzioni di shell possono essere definite in qualsiasi parte di uno script, ma il codice deve sempre precedere il loro utilizzo
- È buona prassi scrivere le funzioni in testa allo script in una sezione ben delimitata dedicata solo alle funzioni

Esecuzione di un Comando

- Distinguamo 2 casi:
- Quando il comando non contiene il carattere '/' la shell tenta di localizzarlo mediante i seguenti passi:

  1) verifica che il comando sia una funzione della shell

  2) verifica che sia un comando incorporato (built-in)

  3) cerca nelle directory specificate nell'ordine in cui sono elencate nella variabile PATH un file eseguibile con il nome specificato
- La shell esegue il primo file eseguibile con il nome specificato che trova
- Quando il comando contiene uno o più '/', la shell esegue il programma corrispondente al pathname specificato

Un semplice esempio

```
#!/bin/bash
rispondesi(){
echo -n " (s/N)? "
read ans
if [ "$ans" == "S" -o "$ans" == "s" ] ; then
        return 0
fi
return 1
}
#Inizio dello script
if [ $# -lt 1 ] ; then
  echo -n Non hai specificato nessun file, vuoi inserirne uno ora
  if rispondesi ; then
    read src
  else
    exit 0
  fi
else
  src=$1
fi
...
```

Parametri

- All'atto dell'invocazione è possibile specificare uno o più parametri di input, con le stesse modalità degli script di shell
- I parametri vengono elencati subito dopo il nome della funzione separati con spazi
- All'interno della funzione sarà possibile accedere ai parametri specificati al momento dell'invocazione mediante le variabili $1, $2, ...
- Analogamente a quanto avviene per gli script `$0` contiene il nome della funzione, `$#` il numero di parametri passati e `$@` la lista completa di parametri

Passaggio parametri

```
#!/bin/bash
# funzione esisteleggibile controlla se il file passato come argomento
# esiste e se leggibile e fornisce due distinti messaggi di errore
esisteleggibile (){
[ $# -eq 1 ] || return 1
if [ ! -e $1 ] ; then
  echo $1 non esiste && return 2
elif [ ! -r $1 ] ; then
  echo $1 non leggibile && return 3
else
  return 0
fi
}
#Inizio script
# Scopo: verifica se i file passati come argomento sono script bash
for file in $@ ; do
  if esisteleggibile $file ; then
    head -n 1 $file | grep -q "^#!/bin/bash"
    if [ "$?" == "0" ] ; then
      echo $file
    fi
  fi
done
```

Parametri e papere

- ATTENZIONE All'interno del codice delle funzione le variabili $1, $2, ... si riferiscono ai parametri passati all'atto dell'invocazione e non a quelli passati sulla linea di comando allo script

Passaggio parametri

```
#!/bin/bash
# funzione esisteleggibile controlla se il file passato come argomento
# esiste e se leggibile e fornisce due distinti messaggi di errore
esisteleggibile (){
[ $# -eq 1 ] || return 1
if [ ! -e $1 ] ; then
  echo $1 non esiste && return 2
elif [ ! -r $1 ] ; then
  echo $1 non leggibile && return 3
else
  return 0
fi
}
#Inizio script
# Scopo: verifica se i file passati come argomento sono script bash
for file in $@ ; do
  if esisteleggibile $file ; then
    head -n 1 $file | grep -q "^#!/bin/bash"
    if [ "$?" == "0" ] ; then
      echo $file
    fi
  fi
done
```

$1 è il primo parametro passato alla funzione

$1 è il primo parametro passato allo script

Comando return ed exit status

- Le funzioni, come i comandi, restituiscono un exit status al termine dell'esecuzione
- Il comando built-in return può essere utilizzato per concludere l'esecuzione di una funzione in qualsiasi punto del codice
- Qualora return venga invocato passandogli un intero compreso tra `0` e 255 come argomento, questi assumerà il valore dell' exit status della funzione
- Se non si specifica nessun valore, return, o si omette lo stesso return, l'exit status di una funzione è quello dell'ultimo comando eseguito
- Si noti che return termina l'esecuzione della funzione mentre exit termina l'esecuzione dello script

Variabili e funzioni

- Le variabili dichiarate all'interno di una funzione sono di default variabili globali e pertanto accessibili anche all'interno dello script

Variabili e funzioni

```
#!/bin/bash
toglinonesistenti()
{
for file in $FILELIST ; do
  if [ -e $file ] ; then
    SHORTFILELIST="$SHORTFILELIST $file"
  fi
done
}
FILELIST=$@
echo $FILELIST
toglinonesistenti
echo $SHORTFILELIST
...
```

Variabili locali

- Può essere utile definire variabili locali accessibili soltanto dalle istruzioni della funzione
- A tal fine è necessario dichiarare esplicitamente queste variabili all'interno della funzione utilizzando il comando built-in local
- Il comando local può essere utilizzato soltanto all'interno di funzioni

Variabili e funzioni

```
#!/bin/bash
toglinonesistenti()
{
local SHORTFILELIST
for file in $FILELIST ; do
  if [ -e $file ] ; then
    SHORTFILELIST="$SHORTFILELIST $file"
  fi
done
FILELIST=$SHORTFILELIST
}
FILELIST=$@
echo $FILELIST
toglinonesistenti
echo $SHORTFILELIST #Stampa una stringa vuota
echo $FILELIST
...
```

Laboratorio di sistemi operativi
A.A. 2010/2011
Gruppo 2
Gennaro Oliva
8
awk

Al Aho

Peter J. Weinberger

Brian W. Kernighan

Caratteristiche di awk

- awk e' un linguaggio di programmazione data-driven che consente di analizzare e elaborare file di testo
- A differenza di quanto accade per i normali linguaggi procedurali con awk si descrivono le informazioni su cui operare e le si associano a determinate istruzioni con cui elaborarle
- awk è anche il comando che esegue l'interprete del linguaggio awk

Storia

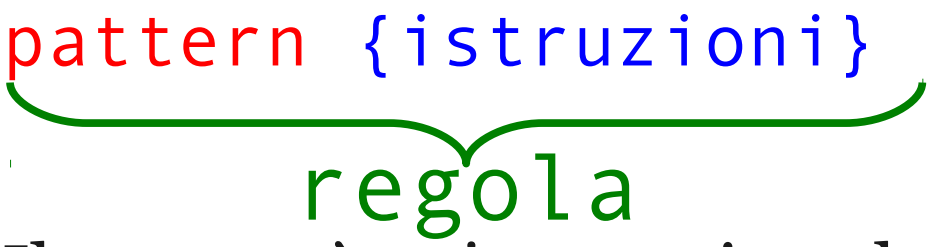
- La prima versione di awk risale al 1977 e fu distribuita con la versione 7 di Unix di AT&T
- Nel 1985 gli autori cominciarono a lavorare ad un'estensione del linguaggio descritta poi nel libro “The AWK Programming Language” del 1988
- La prima versione di awk che supportava il nuovo linguaggio fu distribuita con UNIX System V
- Per evitare confusioni con la vecchia versione incompatibile con la nuova, questa viene talvolta indicata con il termine nawk (new awk)

Implementazioni

- La versione originale, ancora oggi mantenuta da Brian Kernighan, adottò una licenza libera nel 1996
- Le moderne distribuzioni Linux adottano principalmente due implementazioni di awk
- gawk l'implementazione realizzata nell'ambito del progetto gnu
- mawk un'implementazione molto efficiente basata su bytecode
- Esistono anche altre implementazioni ma quella adottata come riferimento per questo corso e mawk

Funzionamento di awk

- Un programma awk è una sequenza di regole



- Il pattern è un'espressione logica che può essere utilizzata per selezionare i dati su cui operare usando la sintassi delle espressioni regolari
- Le istruzioni contenute tra parentesi graffe descrivono le azioni da compiere quando il pattern specificato è un'espressione vera
- In una regola awk si possono omettere sia il pattern che le istruzioni ma non entrambi
  - Se il pattern è omesso le azioni vengono compiute su tutte le linee dell'input
  - Se le istruzioni sono omesse l'azione di default è stampare tutte le linee corrispondenti al pattern

Un semplice programma awk

- Il pattern può essere un'espressione regolare delimitata tra due '/'
- In tal caso le istruzioni corrispondenti vengono eseguite se il testo in esame corrisponde all'espressione regolare
- Esempio:
- Per utilizzare awk come grep:

```
$ awk '/^oliva/ {print}' /etc/passwd
```

- In alternativa essendo la stampa l'azione di default avremmo potuto omettere le istruzioni

```
$ awk '/^oliva/ ' /etc/passwd
```

- Per utilizzare awk come cat:

```
$ awk '{print}' /etc/passwd
```

Programma awk

- La sintassi di un programma awk è la seguente:

```
BEGIN {istruzioni-iniziali}
pattern1 {insieme-istruzioni1}
pattern2 {insieme-istruzioni2}
…
END {istruzioni-finali}
```

- Le regole istruzioni che seguono BEGIN ed END vengono eseguite rispettivamente all'inizio e alla fine del programma
- Le regole sono solitamente poste su linee separate o sulla stessa linea separandole con il simbolo ';'

Esecuzione di awk

- Il comando awk interpreta i programmi scritti nel linguaggio awk e viene invocato utilizzando la sintassi:

  ```
  $ awk 'programma-awk' file1 file2 ...
  ```

  specificando il programma awk esplicitamente sulla linea di comando, opportunamente protetto dagli apici singoli

- Se i file su cui eseguire il programma non vengono specificati sulla linea di comando, l'interprete processa lo standard input

Esecuzione di awk

- Una sintassi alternativa prevede la lettura del programma awk da un file specificato come argomento dell'opzione -f :

  ```
  $ awk -f fileprogram file1 file2…
  ```

- Questo rende possibile scrivere programmi awk eseguibili direttamente dalla shell, scrivendo in testa al file

  ```
  #!/usr/bin/awk -f
  ```

  e rendendoli eseguibili

Caratteri speciali

- Il carattere # si utilizza come per gli script di bash per inserire commenti nel programma
- Tutto quello che segue # fino al carattere di newline viene ignorato dall'interprete
- Esempio:

```
#Inizializziamo le variabili
BEGIN {istruzioni-iniziali}
```

- Il carattere \ viene utilizzato per suddividere un istruzione su più righe e va inserito come ultimo carattere sulla riga da spezzare

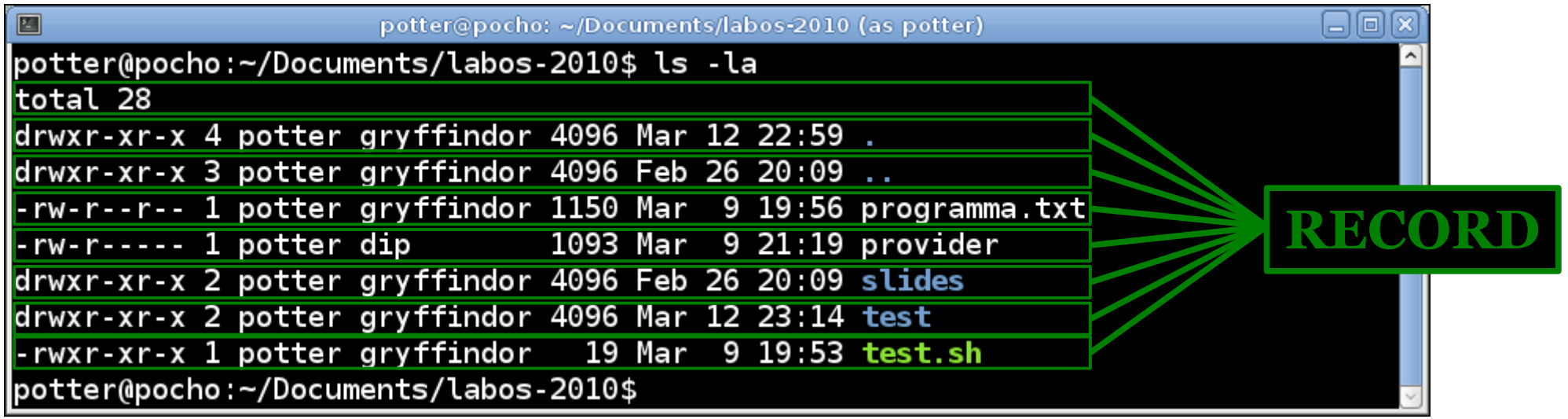
```
pattern {istruzione1 \
istruzione2}
```

record

- L'input viene letto in blocchi chiamati record e viene processato un record alla volta in base alle regole del programma
- I record all'interno di un file sono separati da un carattere (o un'espressione regolare) chiamato record separator
- Il carattere utilizzato come record separator di default è il carattere di new line (a capo)
- In tal caso ogni linea del file costituisce un record

record dell'output di ls -l

- Supponendo di voler analizzare l'output di ls -l mediante awk avremmo che ogni elemento elencato costituisce un record
- Utilizzando mediante pipeline awk ogni linea prodotta in output da ls -l è un record, quindi non solo l'elenco del contenuto della directory ma anche la linea con i blocchi totali

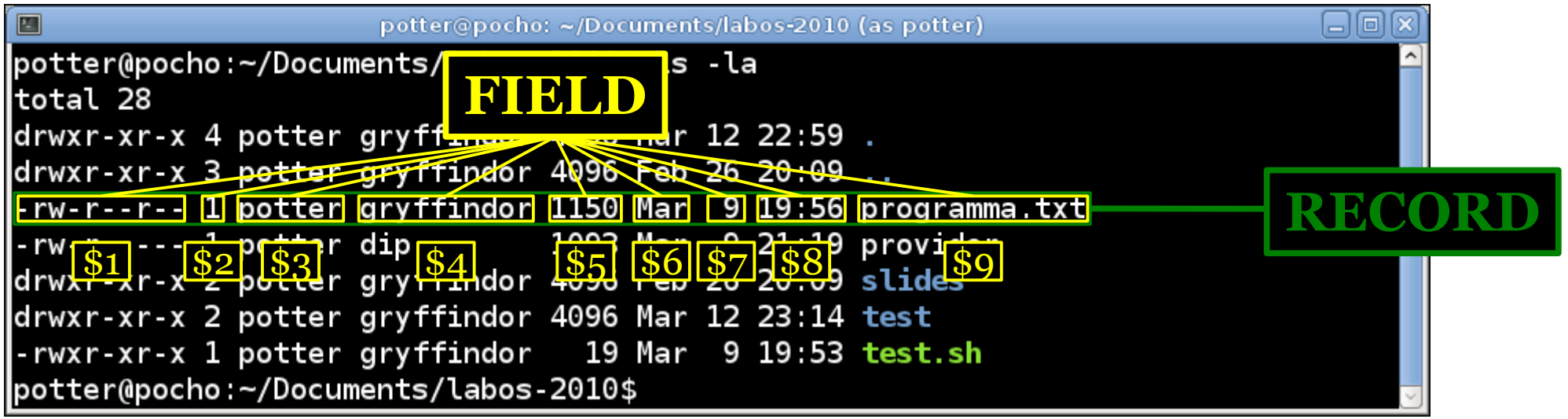


field

- Ogni record viene suddiviso in field (campi)
- I field all'interno dei file sono separati da un carattere (o un'espressione regolare) chiamato field separator
- Di default i campi sono separati da whitespace costituiti da uno o più spazi, tab o newline
- La suddivisione in field rende possibile accedere alle singole componenti di un record
- L'accesso avviene utilizzando il simbolo $ seguito dal numero del field desiderato $1 per il primo $2 per il secondo $11 per l'undicesimo (senza dover delimitare il valore numerico come con bash)
- Diversamente dalla bash `$0` corrisponde all'intero record

field nell'output di ls -l

- I field nell'output ls -l sono, permessi, numero di link, utente, gruppo, dimensione, nome a partire dal secondo record
- Il numero di field di un record può variare all'interno dello stesso input: nel primo record ci sono solo 2 field: “total” e 28, nei restanti record ce ne sono 9



Variabili built-in

- Nel corso dell'esecuzione di un programma, awk conserva ed aggiorna una serie di variabili built-in da cui l'utente può ricavare informazioni sul testo e sul programma
- L'accesso alle variabili in lettura o scrittura **non** necessita dell'utilizzo di simboli particolaricome invece accade per la shell dove in lettura si utilizza il carattere $
- Il carattere $ in awk si utilizza soltanto per accedere ai field del record corrente ($1, $2, ...)

Variabili built-in: NR e FNR

- NR è la variabile che contiene il numero di record processati, viene incrementata ad ogni nuovo record
- Quando vengono specificati più file di input sulla linea di comando, NR conta tutti i record man mano analizzati
- FNR è un contatore per i record del file corrente è che viene azzerato all'inizio di ogni file esaminato
- FILENAME contiene il nome del file correntemente analizzato dall'interprete

Variabili built-in: NF

- NF (number of field) è la variabile che contiene il numero di field del record corrente, viene aggiornata ad ogni nuovo record letto
- Il suo valore può essere utilizzato nell'accesso all'ultimo field anteponendo il simbolo $:
- Ogni record è suddiviso in NF field:
  $1,$2, ..., $NF
- Il simbolo $ può essere seguito da una generica espressione numerica, per accedere al campo desiderato
- Esempio:
  l'espressione $(NF-1) ci fa accedere al penultimo campo

Variabili built-in: RS

- L'interpretazione del carattere di nuova linea come separatore di record, può essere modificata utilizzando la variabile built-in RS (record separator)
- Allo stesso modo per modificare il carattere separatore di field si utilizza la variabile FS (field separator)
- Il contenuto di entrambi le variabili può essere un'espressione regolare: in tal caso i separatori sono tutte le stringhe corrispondenti

Output

- L'istruzione print si visualizza per “stampare” sullo standard output vari elementi: stringhe costanti, valori numerici costanti, il contenuto di variabili, il risultato di espressioni numeriche, ...
- È possibile specificare più di un elemento mediante la sintassi:
  `print elemento1,elemento2`

  oppure:

  `print elemento1 elemento2`
- Quando gli elementi vengono separati con una virgola, awk introduce uno spazio nella visualizzazione, altrimenti vengono concatenati

```
$ awk 'BEGIN {uno=1;due=2;print uno,due,uno due}'
1 2 12
```

Formattazione dell'output

- Analogamente a quanto avviene per l'input è possibile modificare il separatore di default tra due elementi visualizzati da print assegnando un valore alla variabile built-in OFS (output field separator)
- La variabile ORS (output record separator) consente di specificare il separatore tra i record di output prodotti dalle esecuzioni consecutive dell'istruzione print, il cui valore di default è il carattere newline

printf

- Il linguaggio awk fornisce l'istruzione printf come alternativa sofisticata per la stampa su standard output
- In analogia con quanto accade con il linguaggio C la sintassi del comando printf è:
- printf formato, elemento1, elemento2, ...
- Esempio:

```
$awk 'BEGIN {FS=":"}
{printf "%15s\t%s\n",$1,$7}' /etc/passwd
```

- Le variabili OFS e ORS non hanno effetto sulle stampe prodotte da printf

pattern

- In una regola awk
  `pattern {istruzioni}`
  i pattern sono espressioni logiche normalmente usate per selezionare il testo su cui operare
- Le istruzioni corrispondenti vengono eseguite se l'espressione logica è vera (1) mentre non vengono eseguite se risulta falsa (0)
- Esempio:

  ```
  $ awk '$1 == “oliva”  {print}'
  ```

  Stampa tutte le righe il cui primo field è oliva
- Si noti che l'operatore di uguaglianza è == come nel C e non = che invece effettua l'assegnazione

pattern

- Oltre agli operatori relazionali classici (<,>,>=, <=, ==, != è possibile utilizzare l'operatore match ~ con la sintassi:

  expr ~ /regexpr/

- L'operatore restituisce 1 (vero) se l'espressione corrisponde all'espressione regolare e `0` (falso) altrimenti
- Esempio:

  ```
  $1 ~ /d.....r../
  ```

- Quando si specifica soltanto l'espressione regolare nel pattern si sta eseguendo implicitamente

  `$0` ~ /regexpr/

Tipi di dato

- In awk esistono soltanto due tipi di dato: numeri e stringhe
- Le costanti numeriche possono essere interi (-2, 10, ...) o numeri decimali (1.08, 3.14, ...) che possono essere specificati utilizzando la notazione scientifica (-1.1e4, .28E-3, ...)
- Tutti i valori numeri vengono rappresentati internamente in virgola mobile
- Le stringhe costanti vanno sempre racchiuse tra doppi apici
- La scrittura di una stringa può essere suddivisa su più linee utilizzando il carattere \ come ultimo carattere della riga

Variabili

- All'intero di un programma awk possono essere definite variabili con nomi case-sensitive composti da lettere, cifre e dal carattere underscore '_'
- Le variabili possono memorizzare valori numerici e stringhe ed possono variare il tipo di dato memorizzato anche durante l'esecuzione
- Non è necessario dichiarare le variabili
- Di default le variabili sono inizializzate alla stringa vuota che vale zero nel caso di variabili numeriche
- Le variabili built-in possono essere modificate tenendo ben presente che l'interprete può aggiornarle nel corso dell'esecuzione

Assegnazione di una variabile

- L'assegnazione di una variabile viene effettuata la seguente sintassi:

  var=valore

- Nel caso di stringhe è sempre necessario utilizzare i doppi apici
- var=”stringa”
- Le variabili numeriche memorizzano sempre valori floating point e possono essere assegnate utilizzando diverse sintassi

  gol=7
  naplaz=4.3
  spet=6e4 # 60000

Lettura di una variabile

- Diversamente da quanto accade per la shell per utilizzare il valore di una variabile è necessario specificare esclusivamente il nome
- Diversamente da quanto accade per la shell per utilizzare il valore di una variabile è necessario specificare esclusivamente il nome
- Diversamente da quanto accade per la shell per utilizzare il valore di una variabile è necessario specificare esclusivamente il nome
- Diversamente da quanto accade per la shell per utilizzare il valore di una variabile è necessario specificare esclusivamente il nome

Conversione di tipo

- awk converte automaticamente numeri in stringhe e viceversa in base al contesto
- Una stringa non interpretabile come numero equivale a zero
- In generale, il concatenamento di stringhe, impone una trasformazione in stringa, mentre l'uso di operatori aritmetici impone una trasformazione in numero
- Si consideri l'esempio:

```
{decine=1;unita=2;print (decine unita) + 3;}
15
```

- Nel esecuzione del print le variabili numeriche vengono concatenate e quindi convertite in stringhe, generando la stringa concatenata "12"
- Successivamente la stringa viene riconvertita in numero, per la presenza dell'operatore +, che richiede la somma con il numero 3, producendo come risultato finale 15

Operatori aritmetici

- Il linguaggio awk fornisce i seguenti operatori aritmetici

Operazione	Operatore
assegnazione	= += -= *= /= %= ^=
condizionale	? :
or logico	||
and logico	&&
not logico	!
match	~ !~
relazionale	> < >= <= == !=
concatenazione	nessun operatore
aritmetica	+ -
esponenziale	^ oppure **
incremento e decremento	++ -- (sia pre che post)
campo	$
modulo	%