

Shell BASH

Variabili

Redirezione

Variabili

Scrittura/definizione: a=3 (senza spazi)

Lettura: $\{a\}$ o semplicemente \$a

Esempi:

```
> a=3
```

```
> echo $a
```

```
3
```

```
> echo $aa
```

```
> echo  $\{a\}$ a
```

```
3a
```

```
> a=ciao pippo
```

```
bash: pippo: command not found
```

```
> echo "ecco: $a"
```

```
ecco: 3
```

```
> echo 'ecco: $a'
```

```
ecco: $a
```

Variabili predefinite

- PATH percorso di ricerca eseguibili
- USER nome utente
- HOME directory home dell'utente
- PS1 il prompt
- HOSTNAME nome computer
- SHELL la shell corrente
- ...

Per vederle tutte, digitare “set”.

Redirezione

- I programmi dispongono di **3 canali** di comunicazione:
 - standard **input** (codice numerico **0**), per l'input
 - standard **output** (**1**), per l'output
 - standard **error** (**2**), per i messaggi d'errore
- Normalmente:
 - standard input = **tastiera**
 - standard output = standard error = **schermo**

Redirezione

- comando > nomefile redirige lo standard **output** sul file;
se il file non esiste, viene creato;
altrimenti, viene sovrascritto
- comando 2> nomefile redirige lo standard **error** sul file
- comando < nomefile redirige lo standard **input** sul file

Esempi:

```
> echo "ciao" > prova.txt
```

```
> cat prova.txt
```

```
ciao
```

```
> echho "ciao"
```

```
bash: echho: command not found
```

```
> echho "ciao" 2> /dev/null
```

Redirezione

- comando `>> nomefile` redirezione in **append**
- comando `(codiceA)>&(codiceB)` redirige il canale A sul canale B
 - esempio: comando `> file 2>&1`
- Le redirezioni si possono combinare
 - comando `> file1 < file2 2> file3`

Pipe (tubo)

```
comando1 | comando2
```

- L'output di comando1 diventa l'input di comando2
- I due comandi vengono eseguiti in parallelo

Comandi concatenabili

- cat, wc, sort
- less, uniq

Comando cat

- `cat [nome_file]*`
- Scrive su standard output il contenuto di tutti i file, nell'ordine in cui sono specificati
- Senza nomi di file, scrive su standard output tutto quello che riceve da standard input
 - finché standard input non viene chiuso
 - da tastiera, standard input si chiude con Ctrl+d

Comando `wc` (word count)

- `wc [-l] [-w] [-c] [nome_file]*`
- Conta le linee [-l], le parole [-w] oppure i caratteri [-c] dei file passati come argomenti
- Senza opzioni, conta tutti e tre
- Senza nomi di file, conta quello che riceve da standard input

Comando sort

- `sort [-n] [-k num_col] [nome_file]*`
- Ordina le righe dei file passati come argomento
 - scrive il risultato su standard output
 - non modifica i file originali
- opzione `-n`: ordinamento numerico (invece che alfabetico)
- opzione `-k num_col`: ordina in base alla colonna `num_col`
- Senza nomi di file, ordina quello che riceve da standard input

Comando `uniq`

- `uniq`
- Elimina le righe duplicate *consecutive*
- Legge da standard input e scrive su standard output

Comando `less`

- `less [nome_file]`
- Mostra il contenuto di un file, una pagina alla volta
- E' interattivo e ha decine di opzioni
- Senza nome file, legge da standard input
- E' un *pager*

Esercizi

- 1) Elencare i file della directory corrente in ordine alfabetico
- 2) Contare i file della directory corrente che contengono una “z” nel nome
- 3) Scrivere nel file “elenco” l'elenco dei file nella directory corrente, in ordine alfabetico
- 4) Creare un file che si chiami come l'utente corrente
- 5) Creare un file che si chiami come l'host corrente, e che contenga il nome dell'host corrente

Command substitution

- Il pattern `$(comando)` viene sostituito con l'output del comando
- Esempi:
 - `$(ls)` equivale a `*`
 - `$(echo ciao)` equivale a `ciao`
 - `$(cat pippo)` equivale all'intero contenuto del file `pippo`
 - `a=$(ls)` assegna ad `a` l'elenco dei file nella dir corrente
 - `touch "$(date)"` crea un file chiamato come la data attuale

Le sostituzioni

- Variabili: `${variabile}` (*parameter expansion*)
- Caratteri jolly: `* ?` (*filename expansion*)
- Command substitution: `$(comando)`

- Vengono eseguite in quest'ordine
 - esercizio: come si verifica in che ordine vengono eseguite?

Word splitting

- Dopo aver effettuato **parameter expansion** o **command substitution**, la shell effettua la suddivisione in parole
- La variabile **IFS** (internal field separator) definisce i separatori
 - di default, IFS="<space><tab><newline>"
- Come effetto collaterale, il word splitting sostituisce i newline con spazi
 - In una directory con molti file, confrontare l'output di "ls" con quello di "echo \$(ls)"

Word splitting

Consideriamo il programma C **printargs**:

```
#include <stdio.h>
```

```
int main(int argc, char *argv[]) {
```

```
    int i;
```

```
    for (i=0; i<argc ;i++)
```

```
        printf("argv[%d] = \" %s \" \n", i, argv[i]);
```

```
    return 0;
```

```
}
```

> printargs prova questo

argv[0] = "printargs"

argv[1] = "prova"

argv[2] = "questo"

> printargs "prova questo"

argv[0] = "printargs"

argv[1] = "prova questo"

> printargs \$(echo "prova questo")

argv[0] = "printargs"

argv[1] = "prova"

argv[2] = "questo"

Word splitting

```
> temp=$IFS
> IFS="v"
> printargs $(echo "prova questo")
argv[0] = "printargs"
argv[1] = "pro"
argv[2] = "a questo"

> printargs "prova questo"
argv[0] = "printargs"
argv[1] = "prova questo"

> IFS=$temp
```

Esercizi

- 1) Assegnare alla variabile `x` il numero di righe di un file a vostra scelta
- 2) Assegnare alla variabile `x` l'elenco dei file che cominciano con un punto
- 3) Per ogni parola contenuta nel file `pippo.txt`, creare un file con nome uguale a quella parola

Variabili predefinite negli script

- `$1...$9` parametri da riga di comando
- `$#` numero di parametri ricevuti
- `$*` tutti i parametri in una stringa singola
- `$@` tutti i parametri in stringhe separate

Differenza tra \$* e \$@

file *prova*:

```
#!/bin/bash
for x in "$*"; do
    echo "ecco $x"
done
echo qui

for x in "$@"; do
    echo "ecco $x"
done
```

> prova 1 2 3

ecco 1 2 3

qui

ecco 1

ecco 2

ecco 3

Esercizio 1

- Scrivere uno script “**bis**” che prende un comando come argomento e lo esegue due volte
- Esempio: `bis ls -l`
 - esegue due volte “`ls -l`”

Esercizio 2

- Scrivere uno script “**report**” che prende come argomento il nome di una directory ed esamina tutti i file “.txt” in quella directory, producendo un output come il seguente:

```
> report /home/mfaella
```

```
/home/mfaella/a.txt: 23 righe e 368 caratteri
```

```
/home/mfaella/b.txt: 25 righe e 1657 caratteri
```

Riferimenti

- Capitolo 5 di [Introduction to Linux] (redirezione)
- Capitolo 3.2 di [Bash Guide for Beginners] (variabili)