# Sistemi Operativi

AAF - Secondo anno - 3CFU

A.A. 2023/2024

Corso di Laurea in Matematica

# Bash: parametri e altri comandi

Annalisa Massini

Dipartimento di Informatica
Sapienza Università di Roma

# Argomenti trattati

1. Generalità sui comandi
   - Variabili e parametri
2. Comandi sui file
   - Comandi di visualizzazione
   - Comandi per confrontare file
   - Comandi per cercare

## Bash script

# Parametri

# Parametri speciali

- Alcuni **parametri speciali** sono i seguenti:
  - `$0`: il nome dello script, come è stato avviato
  - `$*`: lista di tutti i parametri posizionali (da 1)
  - `$@`: lista di tutti i parametri posizionali (da 1); (con una la differenza per il word splitting)
  - `$#`: numero di parametri posizionali (da 1), separati da uno spazio
  - `$?`: exit status dell'ultimo processo terminato
  - `$!`: pid dell'ultimo processo avviato in background (terminato o no)
  - `$$`: pid della bash (o del parent della bash, se si tratta di una sottoshell)

# Parametri posizionali

- Alcune informazioni sui **parametri posizionali**
  - Si usano solo negli script o nei corpi delle funzioni
  - Un parametro posizionale è definito da una o più cifre numeriche (a eccezione di $0 che ha significato speciale)
  - I parametri posizionali rappresentano gli argomenti forniti al comando: $1 è il primo, $2 è il secondo, e così via
  - $n sta per l'n-esimo argomento dato allo script o alla funzione
  - Se n necessita più di una cifra decimale per essere scritto, allora l'espansione va fatta con le parentesi graffe: ${n}

# Parametri posizionali

- Alcune informazioni sui **parametri posizionali**
  - Con `set {valore}` si possono cambiare i valori dei parametri: se vengono specificati `n` valori, allora il primo viene assegnato a `$1`, il secondo a `$2`, ..., l'`n`-esimo a `$n`
  - Con `shift [n]` si possono cambiare i valori dei parametri
    - Ad esempio con `shift [n]` l'argomento di default vale 1 e i valori dei parametri sono shiftati di 1, cioè `$(2)` avrà il vecchio valore di `$1`, `$(3)` avrà il vecchio valore di `$2` e così via
    - Invece con `shift [n]` i parametri da `$1` a `$n` avranno come valore la stringa vuota (sono non assegnati), mentre `$(n+1)` avrà il vecchio valore di `$1`, `$(n+2)` avrà il vecchio valore di `$2` e così via

## Altri comandi

# Comandi di visualizzazione

# Comando `cat`

- Abbiamo già visto il comando `cat [nomefile]`
- Scrive a schermo il contenuto di `nomefile`
  - funziona bene solo se il file è di testo (e se usa la codifica riconosciuta da `cat`), altrimenti scrive caratteri incomprensibili
  - si può usare `cat` per leggere più di un file alla volta scrivendo `cat file1 file2... filen`:
    - il comando `cat` stamperà il contenuto del primo file, poi del secondo, e così via
    - l'output sarà quindi la concatenazione del contenuto dei file specificati
  - senza argomenti, resta in attesa: se si scrive qualcosa e poi si preme invio, ripete quanto scritto, finché non si preme CTRL+d, che è il carattere EOF (*end-of-file*)

# Comando `tac`

- Il comando `tac [nomefile]` scrive a schermo il contenuto di `nomefile` dall'ultima alla prima riga
  - si può usare `tac` per leggere a rovescio le righe di più di un file alla volta scrivendo `tac file1 file2... filen`:
    - il comando `tac` stampa il contenuto del primo file dall'ultima riga alla prima, poi del secondo sempre dall'ultima riga alla prima, e così via
    - l'output sarà quindi la concatenazione del contenuto dei file specificati ognuno dall'ultima riga alla prima

# Comandi `less` e `more`

- Comandi `less {files}` e `more [-num] [+num] [-d] {files}`
  - come `cat`, ma paginano l'output se è troppo lungo
  - `less` è normalmente usato da `man` per mostrare il manuale
  - differiscono in svariati comportamenti:
    - `less` permette di muoversi sempre sia in avanti che all'indietro, `more` solo se usato senza pipelining (cioè il |)
    - si chiudono premendo `q`, ma per `less` sparisce tutto quello che era scritto, con `more` resta l'ultima schermata
    - `less` pagina sempre, `more` solo se l'output è più grande di una pagina
    - opzioni di `more`: `-num` imposta a `num` il numero di righe in una pagina; `+num` comincia la visualizzazione dalla riga `num`, `-d` mostra una sorta di lista di comandi in basso

# Comando `head`

- Comando `head [-c car] [-n righe] [file...]`
  - al solito, senza argomenti legge da tastiera (una riga per volta)
  - come `cat`, ma stampa solo i primi `car` caratteri o le prime `righe` righe (ciò che è minore) dei file dati
  - senza opzioni stampa 10 righe, senza limiti sui caratteri
  - **Esercizio** scrivere quanto segue su un file, e poi farsi stampare solo la prima riga, ma usando l'opzione `-c`:
    ```
    ciao
    addio
    ```

# Comando `tail`

- Comando `tail [-n righe] [-f] [file...]`
  - al solito, senza argomenti legge da tastiera (ma questa volta occorre premere CTRL+d alla fine dell'input)
  - come `cat`, ma stampa solo le ultime `righe` dei file dati
  - con `-f`, aggiorna di continuo la stampa: utile se si vuole leggere un file cui vengono continuamente aggiunti dati (ad esempio, come risultato di una qualche computazione, *mentre* la computazione stessa è in esecuzione)

# Comandi `head` e `tail`

- **Esercizio** trovare il modo per andare avanti ed indietro di $k$ righe e di $k$ pagine con `more` e `less`
- **Esercizio** creare un file con `gedit` (lanciato in background), scriverci dentro almeno 4-5 righe e poi salvare; tornare sulla shell ed eseguire `tail nomefile`. Poi aggiungere qualche altra riga, ed eseguire nuovamente `tail nomefile`. Effettuare nuovamente questi passaggi, ma eseguendo stavolta `tail -f nomefile`. È sufficiente scrivere le modifiche sul file, affinché vengano visualizzate dal `tail`?

## Comandi sui file

# Comandi per confrontare file

# Comando `diff`

- Il comando `diff [-i] [-b] [-r] [-q] file1 file2` elenca **tutte** le differenze
  - le differenze vengono mostrate riga per riga
  - `-b`: ignora le differenze, se consistono solo in un numero diverso di spazi
  - `-i`: ignora le differenze, se consistono solo nel *case* diverso (minuscolo vs. maiuscolo)
  - `-r`: confronta ricorsivamente 2 directory, confrontando con `diff` i file che hanno lo stesso percorso relativo e segnalando quali file sono presenti in una sola delle 2 directory
  - `-q`: dice solo se i file sono diversi, senza elencare le differenze

# Comando `cmp`

- Il comando `cmp [-b] file1 file2` confronta i 2 file dati
  - `cmp` si limita a trovare la prima occorrenza di un byte in cui differiscono
  - con `-b`, stampa anche tale byte, sia in ottale che nella versione stampabile (se possibile)

# Comandi `cmp` e `diff`

- **Esercizio**
  - Creare un file e scriverci dentro qualche riga
  - Copiarlo in un altro file, farci delle modifiche e poi confrontare i 2 file sia con `diff` che con `cmp`
  - Rifare la stessa cosa, ma con directory al posto dei file; modificare alcuni file e poi eseguire `diff` sia con che senza `-r`

## Comandi sui file

# Comandi per cercare

# Il comando `find`

- Comando `find [opzioni] [directory] [condizione] [azione]`: serve per la ricerca di file
  - La ricerca avviene nella directory specificata e nelle sue sottodirectory, oppure nella cwd se la directory non è specificata
  - Criteri di ricerca tipici sono: il nome del file (-name nomefile [suffisso]), il nome utente (-user nomeutente), le dimensioni del file (-size), l'accesso indicato in giorni (-atime [+-]n) o la modifica indicata in giorni (-mtime [+-]n)
  - Esempio: `find /mydir -name "*.odt" -mtime -3 -size +20k`

# Il comando grep

- Comando `grep [opzioni] stringa [file]`: serve per cercare la stringa specificata in un file di testo
- grep - acronimo di global regular expression print
- La stringa è una sequenza di uno o più caratteri (lettera singola, parola o frase) e può includere spazi, segni di interpunzione e caratteri di controllo
- Solitamente grep viene utilizzato su tutti i file nella cartella corrente
- `-i` ignora le differenze tra lettere maiuscole e minuscole
- `-n` in ogni linea del risultato viene mostrato il numero di riga del file
- `-r` consente una ricerca ricorsiva nelle sottocartelle

# Il comando grep

## Esercizi

- Creare un file di testo contenente nomi cognomi e numeri di telefono e cercare il numero di qualcuno
- Usare nomi parzialmente specificati (esempio: arc per Marco)
- Cercare ignorando la differenza tra maiuscole e minuscole
- Richiedere il numero di riga