UNIX

Introduzione

Come nasce il sistema operativo UNIX?
UNIX viene creato il 1 gennaio 1980. La motivazione dietro la sua creazione era quella di eseguire un programma denominato "Space Travel". Questo software simulava con precisione i movimenti celesti, compresi quelli del sole e dei pianeti, insieme al percorso di una navicella spaziale capace di atterrare in varie località.
Cosa significa oggi UNIX? Linux è UNIX?
Oggi con UNIX vengono definiti i sistemi operativi che rispettano determinate specifiche. Linux è un sistema UNIX-like, che è differente.

Rappresentazione dei dati

Come vengono rappresentati in memoria gli interi con e senza segno?
Con segno il bit più significativo rappresenta il segno (0 positivo, 1 negativo), i bit rimanenti il numero in valore assoluto. Senza segno tutti i bit rappresentano il numero non negativo.
Big endian/little endian, cosa sono?
Sono due convenzioni usate per definire l'ordine con cui vengono memorizzati i byte costituenti un dato da immagazzinare. `Big endian` : dal byte più significativo al meno. `Little endian` : dal byte meno significativo al più.
A cosa servono i numeri ottali o esadecimali?
Sono rappresentazioni usate dai programmatori per comodità, non sono effettivamente usate dal computer.
Le operazioni & (and), | (or), ~ (not), ^ (xor): come funzionano?
- `&` fa AND bit a bit tra due operandi
- `|` fa OR bit a bit tra due operandi
- `~` fa NOT bit a bit di un operando
- `^` fa XOR bit a bit tra due operandi

Come si fa ad "accendere" bit?
Si fa OR bit a bit con un valore che ha 0 nei bit da lasciare invariati ed 1 nei bit da "accendere".
Come si fa a "spegnere" bit?
Si fa AND bit a bit con un valore che ha 0 nei bit da "spegnere" ed 1 nei bit da lasciare invariati.
Dati A e B come si fa a controllare se alcuni bit di A sono accesi in B?
Si fa AND bit a bit tra A e B, i bit a 1 nel risultato sono comuni ad entrambi.
Dati A e B come si fa a controllare se tutti i bit di A sono accesi in B?
Si fa AND bit a bit tra A e B, poi si confronta il risultato con un uguaglianza rispetto ad A.
`(A & B) == A`
Come si fa a considerare solo alcuni bit di un intero (mascheramento)?
Si crea una maschera binaria in cui i bit di interesse sono settati a 1 e tutti gli altri a 0. Si fa poi AND bit a bit tra l'intero e la maschera ottenendo così solo i bit interessati.
Come si può usare il mascheramento per fare l'operazione modulo di una potenza di 2?
`x % (2^n)` è equivalente a `x & ((1 << n) - 1)`

BASH base

Qual è la sintassi tipica dei comandi UNIX?
comando [opzioni] [target del comando]
Le opzioni generalmente iniziano per `-` o `--`.
Cosa è un file system?
È una struttura dati che si occupa di archiviare e gestire file su un computer.
UNIX usa il file system come strumento di naming. Cosa significa?
Strumento di naming è il metodo utilizzato per assegnare nomi univoci a entità, risorse o oggetti all'interno di un sistema. Questo significa che il nome di un file o di una directory è determinato dalla sua posizione gerarchica all'interno del file system stesso.
Cosa sono la home directory e la current working directory?
`home directory` : directory speciale assegnata ad ogni utente del sistema.
`current working directory` : directory attualmente attiva all'interno della quale un utente o un processo sta lavorando.
Qual è la struttura tipica del file system di UNIX?
La struttura del file system UNIX è ad albero gerarchico. La `Root Directory` (indicata con `/`) rappresenta il punto di partenza della gerarchia. Tutte le altre directory e file sono contenuti all'interno di essa. Sotto-directory principali:
- `/bin` : programmi eseguibili (binari) fondamentali necessari per il funzionamento del sistema, come comandi di base del sistema operativo.
- `/boot` : file di avvio del sistema, inclusi i kernel del sistema operativo e i file necessari per il processo di avvio.
- `/dev` : file speciali che rappresentano dispositivi hardware o software, come dischi, tastiere, mouse e altre periferiche.
- `/etc` : file di configurazione di sistema e script di avvio.
- `/home` : directory principale per le home directory degli utenti.
- `/lib` : librerie condivise utilizzate dai programmi del sistema.
- `/root` : home directory dell'amministratore di sistema (root).
- `/tmp` : utilizzata per archiviare file temporanei creati dai processi e dagli utenti.
- `/usr` : contiene la maggior parte dei programmi di sistema, librerie e file di dati.
- `/var` : contiene dati variabili del sistema, come log di sistema, file di stampa, e-mail in coda ...
Cosa sono i Pathname assoluti e relativi
Un pathname è una stringa di caratteri utilizzata per specificare la posizione di un file o di una directory all'interno del file system.
`assoluto` specifica la posizione completa a partire dalla root directory.
`relativo` specifica la posizione rispetto alla current working directory (fa uso di . e ..).
Cosa fanno i comandi pwd, cd, ls, echo, date, cat, more, less?
- `pwd` : stampa la directory attuale.
- `cd` : cambia directory.
- `ls` : lista i file/directory all'interno della directory corrente.
- `echo` : ripete il parametro.
- `date` : visualizza data e ora esatta.
- `cat` : concatena il contenuto di due file in output. Se non si specifica il secondo file, mostra il contenuto del primo.
- `more` : mostra l'intero contenuto di un file.

- `less` : mostra il contenuto di un file in pagine.

Cosa fanno i comandi touch, cp, mv, mkdir, rm, rmdir?

- `touch` : crea un file.
- `cp` : copia uno o più file in una directory.
- `mv` : sposta uno o più file in una directory.
- `mkdir` : crea una directory.
- `rm` : elimina un file o una directory.
- `rmdir` : rimuove tutte le cartelle vuote.

A cosa serve il comando man? Quali sono le sezioni di man?

Consente di consultare il manuale di ogni comando, funzioni di libreria, system call, strumenti di amministrazione. Le sezioni di man sono NAME, SYNOPSIS, DESCRIPTION, OPTIONS, EXAMPLES e SEE ALSO.

Cosa fanno i comandi file, whoami, groups

- `file` : indica il tipo di file.
- `whoami` : identifica l'utente attuale.
- `groups` : indica i gruppi di cui fa parte l'utente.

Quale è il significato dei singoli elementi delle righe ottenute con "ls -l"?

Primo carattere d è una cartella, - è un file. Poi i permessi in tre gruppi di tre lettere (r = read, w = write, x = execute): il primo è per il proprietario (u), il secondo per chi fa parte del gruppo del file (g), il terzo per tutti gli altri (o). Segue il numero di nomi associati al file, l'utente proprietario, il gruppo proprietario, la dimensione, data e ora dell'ultima modifica ed il nome.

Come si cambiano i permessi di un file o una directory?

Si usa il comando `chmod` . Sintassi: `[utenti (u, g, o)] +/- [permesso (r, w, x)]` .
Una sintassi alternativa è usare 3 cifre ottali che rappresentano i permessi. `chmod 755 foo.txt`

Cambiare utente o gruppo proprietario di un file

- `chown` : cambia l'utente proprietario di un file/dir.
- `chgrp` : cambia il gruppo di appartenenza di un file/dir.

Link fisici e simbolici: cosa sono e quali sono i comandi per crearli?

Servono per creare collegamenti tra file o directory all'interno del file system.
`link fisico` : contiene gli stessi dati e permessi del file originale. Se viene eliminato il file originale, il link rimane comunque intatto. Si crea con `ln -l`
`link simbolico` : è un collegamento al file originale. Se viene eliminato il file originale, si rompe il link simbolico. Si crea con `ln -s`

Esecuzione in foregound e background, cosa significa?

Sono due modalità di esecuzione dei processi in un sistema operativo.
`foreground` : il processo è in primo piano e blocca il prompt del terminale fino a quando non è terminato o non viene messo in pausa manualmente.
`background` : il processo viene avviato in secondo piano senza bloccare il prompt. Si fa con `nome-comando &`

Job control

Permette di portare i processi da background a foreground e viceversa. `jobs` lista i processi in background. `fg %n` porta il processo n-esimo da background a foreground.

Cosa fanno i comandi find, grep?

- `find` : cerca file in una directory.
- `grep` : cerca un testo in un file e mostra le righe del file che lo contengono.

du, df
- `du` : mostra lo spazio occupato sui dischi.
- `df` : mostra lo spazio libero nei dischi.

pid ppid
`pid` è l'identificatore del processo, `ppid` è l'identificatore del processo padre (parent pid).

nice, renice, kill
- `nice` : esegue un comando con una priorità statica (detta nice number) definita.
- `renice` : cambia la priorità di un processo in esecuzione.
- `kill` : termina un processo.

ps, top
- `ps` : riporta lo stato dei processi attivi nel sistema.
- `top` : mostra tutti i processi attivi nel sistema.

Shell

La shell è un programma posto tra utente e SO. Esistono diversi tipi di shell ma le più diffuse in ambiente Linux sono `bash` o `Bourne shell (sh)` . Il software libero permette di usare la shell che ti pare. `echo $SHELL` per vedere quella attualmente in uso.
Alcuni comandi sono built-in nella shell e vengono eseguiti direttamente (tipo cd o echo), altri sono esterni e vanno caricati prima da un file (ad esempio ls si trova in /bin/ls).

I file standard: stdin, stdout, stderr. Cosa sono e come si usano?
Sono 3 canali di comunicazione usati per gestire input e output dei programmi.
- `stdin` è il file di standard input e rappresenta il canale attraverso il quale i programmi leggono l'input. Solitamente è associato alla tastiera dell'utente.
- `stdout` è il file di standard output e rappresenta il canale attraverso il quale i programmi scrivono messaggi di output. Solitamente è associato al terminale dell'utente.
- `stderr` è il file di standard error e rappresenta il canale attraverso il quale i programmi inviano messaggi di errore. Solitamente è associato al terminale dell'utente.

Cosa è e a cosa serve la ridirezione?
Consiste nel cambiare la destinazione di stdin o stdout di un programma da e verso file anzichè utente o terminale. Questo permette di manipolare i dati in modo più flessibile.
- `stdin` usa il simbolo `<` (es: /programma < f_in legge l'input da f_in)
- `stdout` usa il simbolo `>` (es: /programma > f_out scrive l'output in f_out)
- `stderr` usa il simbolo `2>` (es: /programma 2> f_err scrive gli errori in f_err)

Cosa è e a cosa serve la pipe?
È un carattere speciale (`|`) usato per concatenare due o più comandi in modo che l'output di un comando venga utilizzato come input per un comando successivo.

Simboli speciali nella scrittura dei pathname (wildcards): quali sono e cosa significano?
Sono caratteri speciali usati per simulare pattern nei nomi dei file e delle directory.
- `*` rappresenta zero o più caratteri qualunque.
- `?` rappresenta esattamente 1 carattere qualsiasi.
- `[ ]` consente di specificare un set di caratteri validi (es: [0-9] corrisponde a qualsiasi cifra).
- `[! ]` consente di specificare un set di caratteri non validi.

Command substitution
Un comando racchiuso tra contro-apici viene eseguito, ed il suo output viene sostituito al posto del comando. La forma preferita è però $(nome_comando). `echo Data odierna: $(date)`

Sequenze
Il carattere `;` serve per eseguire comandi sequenzialmente (sequenza non condizionale)
Sequenze condizionali
- `&&` : il comando successivo viene eseguito solo se il primo è andato a buon fine.
- `||` : se il primo comando va a buon fine il secondo non viene nemmeno eseguito (viene eseguito solo se il primo fallisce).
Subshell, ()
Comandi raggruppati dentro le parentesi tonde `()` vengono eseguiti insieme in una subshell. Condividono gli stessi stdin, stdout e stderr. Viene creata una subshell anche quando si esegue uno script o quando viene eseguito un processo in background.
Quoting
Apici singoli `'` inibiscono wildcard, command substitution e variable substitution. Apici doppi `"` disattivano solo le wildcard.
Variabili shell e variabili di ambiente (environment)
Le `variabili shell` (o locali) sono usate per computazioni interne a script, non vengono ereditate dalle subshell create. Le `variabili di ambiente` sono utilizzate per comunicare tra shell e subshell, naturalmente vengono ereditate. Entrambe le variabili contengono dati di tipo stringa. Ogni shell ha alcune variabili di ambiente inizializzate di default, comando `env` per vedere l'elenco.
Per assegnare un valore ad una variabile locale (e crearla se non esiste) usare `nome_var=valore`, se il valore contiene spazi usare i doppi apici. Per trasformarla in una variabile di ambiente il comando è `export nome_var_locale`.
Per usare una variabile si scrive `$nome_var` che abbrevia `${nome_var}` (a volte è necessario intero).
`readonly var` per rendere la variabile *var* non modificabile.

Shell scripting

Script e sharp-bang
Uno script è un file di testo, contenente comandi, a cui è assegnato il permesso di esecuzione `x`.
Per eseguire lo script in una shell predefinita inserire come prima riga `#!interpreter_path`. Per usare bash: `#!/bin/bash`, per usare la shell che invoca lo script mettere solo `#`.
Di base `#` è il carattere per inserire commenti su singola linea.
Variabili built-in per scripting
- `$$` : l'identificatore di processo della shell
- `$0` : il nome dello shell script
- `$n o ${n}` : n-esimo argomento della linea di comando (graffe per numeri a più cifre)
- `$*` : lista tutti gli argomenti della linea di comando
- `$#` : numero di argomenti sulla linea di comando
- `$?` : valore di uscita dell'ultimo comando eseguito
Here is document <<
Sintassi: `command << limit`
Esegue *command* usando come standard input tutto il testo successivo al comando fino alla riga che inizia con la parola *limit*. `<<` esegue variable substitution, `<</` no.
expr
Comando `expr [expression]` per valutare un'espressione. Tutti i componenti dell'espressione

devono essere separati da spazi, tutti i metacaratteri (ad es. *) devono essere prefissi da un backslash.
Operatori: `+ - * / %, = != < > <= >=, & | !, (), match substr length index`

Exit status

Valore di ritorno di un processo quando termina. 0 in caso di success (true), qualunque altro valore in caso di failure (false).
`exit s` per terminare lo script con exit status s.

test [] [[]]

Tre comandi equivalenti per testare un'espressione: exit code 0 se è true, 1 se è false. `[[ expression ]]` è preferibile a `[ expression ]`. Si possono usare gli operatori di confronto e logici come in C, inoltre:
- Controlli su stringhe: `s1 = s2` equal, `s1 != s2` not equal, `-z str` empty, `-n str` not empty.
- Controlli su file `-... filename`: `-d` directory exists, `-f` file exists, `-r` exists and readable, `-w` exists and writable, `-x` exists and executable, ...
`test expression` è la versione originale che usa le opzioni anzichè gli operatori in segno.

if/then/elif/fi

```
if condition
then comandi_se_true
elif altra_condition
then comandi_se_true
else comandi_se_false
fi
```

case..in/esac

I pattern possono contenere wildcard.

```
case string_expr in
pattern1 )
comandi_se_matcha_pattern1
;;
pattern2 )
comandi_se_matcha_pattern2
;;
esac
```

while..do/done

Si possono usare `break` e `continue` all'interno del corpo.

```
while condition
do corpo_di_comandi
done
```

for..in/do/done

Equivalente al ciclo for each. Se la clausola `in` è omessa, `$*` viene utilizzato al suo posto. Si possono usare `break` e `continue` all'interno del corpo.

```
for variabile in lista
do corpo_di_comandi
done
```

Comandi built-in echo, printf, read

- `echo` : stampa i suoi argomenti su stdout. `-n` per togliere il new line dopo la stampa.
- `printf` : versione avanzata di `echo` che permette di formattare l'output.
- `read` : legge l'input da tastiera e lo salva in una variabile. `-nN` per accettare N caratteri in input.

let (bash only)

`let` esegue operazioni aritmetiche sulle variabili. Funziona come una versione più semplice di `expr` .

unset, true, false, type

- `unset` : cancella il contenuto di una variabile.
- `true` : ritorna sempre exit status 0.
- `false` : ritorna sempre exit status 1.
- `type` : dice se il comando è un built-in, una keyword o un comando esterno.

source (or dot)

Esegue uno script senza aprire una subshell. `source nome_script`

Alias

Un alias è una scorciatoia per abbreviare lunghe sequenze di comandi. Con `alias nome="comandi"` si crea, con `unalias nome` si distrugge.

Cosa è history

`history` permette di vedere l'elenco dei comandi eseguiti in precedenza. `-c` per cancellare la cronologia.

Directory stack

La shell Bash permette di creare e manipolare una pila di directory, utile per visitare un albero di directory.

- `dirs` : stampa il contenuto del directory stack.
- `pushd dir_name` : aggiunge *dir_name* in cima allo stack e si sposta al suo interno.
- `popd` : rimuove l'elemento in cima allo stack e si sposta nella nuova directory in cima.
- `$DIRSTACK` : variabile che contiene la directory in cima allo stack.

Comandi cat, tac, rev

- `cat` : stampa i file su stdout. Utilizzato con redirection serve a concatenare file.
- `tac` : stampa le righe di un file dall'ultima alla prima.
- `rev` : stampa le linee di un file in ordine inverso partendo dalla prima.

find

`find pathlist expression` analizza ricorsivamente i path contenuti in *pathlist* e applica *expression* ad ogni file.

- `-name pattern` per agire solo sui file il cui nome fa match con *pattern* (può usare wildcard).
- `-perm permission` per agire solo sui file con i permessi *permission*.
- `-print` per stampare il pathname del file.
- `-ls` per stampare gli attributi del file.
- `-user username` o `-uid userid` : per agire solo sui file il cui proprietario è *username/userid*.
- `-group groupname` o `-gid groupid` : per agire solo sui file parte del gruppo *groupname/groupid*.
- `-type ...` : per selezionare i file in base al tipo.

Per legare le opzioni si possono usare gli operatori logici `-and` , `-or` e `-not` .

xargs

`xargs command` legge una lista di argomenti dallo standard input, delimitati da spazi oppure newline, ed esegue *command* passandogli la suddetta lista di argomenti.

head/tail/cut

- `head -n file` : lista le prime *n* righe di *file*.
- `tail -n file` : lista le ultime *n* righe di *file*.
- `cut` : permette di estrarre campi dai file. `-d` per specificare il carattere limite, `-f` per specificare quali campi stampare.

uniq/sort

- `uniq` : rimuove le linee duplicate consecutive dallo stdin.
- `sort` : ordina lo stdin linea per linea. `-g` per ordinare in modo numerico sul primo campo.

wc/tr

- `wc` (word count): conta linee `-l` , parole `-w` o caratteri `-m` .
- `tr` : consente la conversione di caratteri seguendo un insieme di regole definite dall'utente.

cmp e diff

- `cmp` : ritorna true se due file sono uguali. Se sono diversi ritorna false e stampa la prima riga con differenze. `-s` per non stampare nulla.
- `diff` : se usato con due file ritorna true se sono uguali, altrimenti ritorna false e stampa l'elenco delle differenze. Se usato con due directory mostra i file presenti in una ma non nell'altra.

basename e dirname

Sono funzioni su stringhe, non agiscono sul file effettivo.

- `basename` : rimuove l'informazione del path da un pathname.
- `dirname` : rimuove l'informazione del nome di file da un pathname.

time

`time command` esegue *command* e stampa una serie di statistiche sul tempo impiegato.

tar

Uno dei tanti comandi per comprimere e decomprimere file.

- Archiviazione: `tar zcvf archive_name files`
- Estrazione: `tar zxvf archive_name`

sed/awk

`sed` e `awk` sono comandi che effettuano la scansione di uno o più file (o dello stdin) ed eseguono un'azione su tutte le righe che soddisfano una particolare condizione.

Funzioni shell

Una funzione è un blocco di codice che implementa una funzionalità ripetitiva, permette di organizzare al meglio il codice.

```
# dichiarazione (prima dell'invocazione)
function_name() {
  comandi
}

# invocazione
function_name parametri
```

La funzione si riferisce ai parametri in modo posizionale con $1, $2, ..., ${n}.
Una funzione può restituire un valore (exit status) che di default è uguale all'exit status dell'ultimo comando eseguito. Per specificare un valore di ritorno *x* si usa `return x`. Per ritornare stringhe generali si utilizza la variabile `$REPLY`.