

Sistemi Operativi

Leonardo Mengozzi

Titoletti indice link a rispettive sezioni, in alto a sinistra "[←Indice](#)" link a pagina Indice.

Contents

1 Batch	1
Comandi V.Ambiente	1
Comandi File speciali	1
Comandi Directory	1
Comandi Scripting	2
Comandi Privilegi	2
Comandi Subshell	2
Comandi Vari	3
2 Programmazione Concorrente	3
Comandi Multi-processo	3
Comandi PIPE	4
Comandi Multi-thread	5

1 Batch

V.Ambiente	Descrizione
PATH	Modificabile, sequenza di percorsi assoluti, divisi da ":", di directory contenenti eseguibili (lanciabili senza digitare path). Ricerca secondo ordine specificato in PATH, si ferma a primo eseguibile con nome uguale. Eventuale ErrorNotFoutd. Altre variabili d'ambiente: \$HOME, \$USER, \$SHELL, \$TERM.
env	Visualizza l'elenco delle variabili d'ambiente.

File speciali	Descrizione
/etc/passwd	Righe sono info ogni utente divise da ":".
/etc/shadow	Righe sono password utente codificate.
/etc/group	Righe sono info ogni gruppo divise da ":".
/usr/bin/passwd	Cambia la pass utente.

Directory	Descrizione
<code>cd percorso</code>	Sposta logicamente in una diversa directory, secondo un path assoluto o relativo.
<code>ls</code>	Visualizza i files/direcotry contenuti nella directory corrente.
<code>ls -a</code>	ls ma mostra anche file nascosti (anche <code>.</code> , <code>..</code>).
<code>ls -l</code>	ls ma mostra più informazioni sui files.
<code>ls nomefile</code>	Dice se il file esiste o no, e con i parametri opzionali da altre info solo per quel file.
<code>pwd</code>	Visualizza il percorso assoluto, da <code>/</code> fino alla directory corrente.
<code>touch nomeFile.estensione</code>	Crea un file vuoto nella dyrectory corrente.

Scripting	Descrizione
<code>echo testo</code>	Visualizza a video la sequenza di caratteri passata fino al primo "INVIO".
<code>\</code>	Disabla interpretazione per il carattere successivo, andata a capo, permettendo di stamparlo.
<code>echo "testo"</code>	Come echo, ma disabilita interpretazione caratteri speciali e andate a capo.
<code>;</code>	Separatore di più comandi lanciati durante la stessa digitazione.
<code>nome=valore</code>	Simboli con nome e valore, stringa modificabile, alfanumerici casesensitive. No spazi prima o dopo "=". Sono d'ambiente o ex-novo locali. Solo la bash in cui sono create le variabili le può usare. I programmi lanciati dalla bash hanno una speudocopia della bash.
<code>export nomevar nomevar=valore</code>	Variabile d'ambiente. Un shell figlia riceve una copia modificabile che non influenza variabile d'ambiente del padre.
<code>unset nomevariabile</code>	Elimina una variabile esistente (vuota o no). Quotare ("...") sempre variabili per evitare errori con variabili vuote o inesistenti.
<code>\${nomeVari}</code>	Fa sostituire il nome della variabile con il valore. graffe opzionali se nome variabile seguito da uno spazio.
<code>#...</code>	Commeto.
<code>#!...</code>	Se indicato nella prima riga indica quale interprete deve eseguire lo script. Se non specificato usato quello corrente.
<code>comando1 comando2</code>	pipe (speudo-file temporaneo): collega automaticamente l'output di un comando all'input di un altro. Unidirezionale.

Privilegi	Descrizione
<code>chmod u+x script.sh</code>	Modifica permessi file mediante formato numerico: u+x terna 0-7. Ogni numero è la somma dei valori associati ai permessi di r(4), w(2), x/s(1). Ordine: proprietario, gruppo, altri utenti. Può diventare un quartetto aggiungendo per primo l'identificatore numero dei privilegi di <i>setuid</i> , <i>setgid</i> , <i>sticky bit</i> .
<code>chgrp ???</code>	Modifica il gruppo di appartenenza di un file.
<code>chown newOwner nameFile</code>	Modifica il proprietario (e anche gruppo) di un file.
<code>ls -al nomeFile.estensione</code>	ls che mostra permessi, anche dei file nascosti. Interpretazione: 1°carattere tipo file (- file, d dyrectory, c collegamento seriale, b device a blocchi), 9 caratteri successivi terzine di permessi (r read, w write, x/s execute) per proprietario, gruppo, altri utenti.
<code>whoami</code>	Dice all'utente corrente le sue informazioni.
<code>sudo comando</code>	fa eseguire il comando come amministratore, può essere chiesta userPass. Solo utenti gruppo sudo (gestito dall'admin) possono usarlo.

Subshell	Descrizione
bash	Crea una bash figlia, eredita dal padre: dir. corrente, copia variabili d'ambiente. Non sono ereditate le variabili locali. Creata in automatico da: comandi raggruppati, script, processi in background. Nota: i built-in eseguiti in shell corrente/padre.
bash -c <i>script.sh</i>	Esegue in una subshell, con l'interprete opzionalmente indicato, lo script specificato.
var=val comando	Scrivendo le assegnazioni prima dell'esecuzione di un comando si creano delle var. d'ambiente solo per l'imminente subshell. Non saranno ereditate da successive subshell.
. source <i>script.sh</i>	Esegue lo script nella shell corrente. Utile a impostare/modificare variabili shell. Ignorata prima riga opzionale e eseguito con interprete corrente.
exit	Termina bash corrente, elimina l'ambiente e sale alla padre.
top	Mostra in tempo reale processi in esecuzione e risorse di sistema usate.
ps	Mostra i processi in esecuzione. Con l'opzione -all vedo più informazioni (PID, PPID, ecc).
set	Visualizza tutte variabili della shell corrente.

Vari	Descrizione
./ <i>comando</i>	Esegue comando presente nella directory corrente (percorso relativo). Alternativa è inserire il percorso relativo o aggiungere il percorso assoluto alla variabile PATH.
strace <i>comando</i>	Dice la sistem-call usata.
clear	Pulisce la CLI. Non modifica variabili create.
which <i>comando</i>	Cerca in PATH il comando e se lo trova mostra il path in cui si trova.
cat <i>nomeFile.estensione</i>	Visualizza il contenuto del file.
lsmod	Elenca tutti i moduli attivi.
modinfo <i>nomeModulo</i>	Dice le informazioni sul modulo specificato.
sudo modprobe <i>nomeModulo</i>	Carica il modulo specificato.
history	Visualizza comandi, numerati, precedentemente eseguiti, anche da shell precedentemente chiuse. Con !NUMERO lancio il comando corrispondente nell'elenco di history. Con !STRINGA lancio il comando più recente che corrisponde alla stringa.

Directory "proc" info sui processi memorizzati con una dir. per ogni processo attivo, create e cancellate continuamente.

2 Programmazione Concorrente

Inclusioni necessarie:

- **#include <sys/types.h>** Definire tipi di dato speciali usati nelle chiamate di sistema.
- **#include <unistd.h>** Include funzioni e costanti del sistema operativo Linux/Unix.

Multi-processo	Descrizione
fork()	Crea un processo figlio. Condivide codice successivo alla fork e possiede una copia dei dati. Restituisce il pid del figlio al padre ($PID_F > 0$) e 0 al figlio ($PID_F = 0$) o un codice d'errore ($\neq 0$).
exit(0)	Termina un processo restituendo lo stato indicato come parametro (0 stato successo). Se padre termina prima del figlio non si possono liberare le risorse.
wait(NULL)	Fa attendere al processo la terminazione del processo figlio. Restituisce il pid del processo terminato.
wait(pidProcesso)	Fa attendere al processo corrente uno specifico processo.
waitpid(pidProcesso, &status, statoAtteso)	Fa aspettare un thread specifico con uno stato specifico.
exec	Sostituisce codice e dati a un processo. Non crea processi figlio.

Un processo possiede: Il proprio PID (gestito dall'os), il PID del processo figlio o 0 e il pid del processo "iziale" (gestito da descrittore di file).

Struttura base programma multi-processo:

```
int main() {
    pid_t pid;

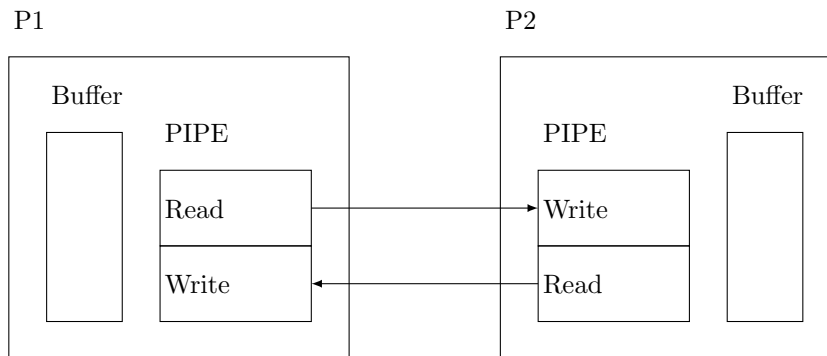
    pid = fork();

    if (pid < 0) { "gestione errore" }
    else if (pid == 0) { "processo figlio" }
    else { "processo padre" }
}
```

Ogni processo tiene referenza di un unico figlio, anche se ne può creare diversi. Quindi dopo ogni fork è buona cosa separare i flussi dei due processi.

PIPE	Descrizione
pipe(varPipe[2] : int) : int	Crea una pipe unidirezionale. Su varPipe[0] si leggerà. Su varPipe[1] si scriverà. Restituisce -1 in caso di errore, 0 se creazione avviene con successo.
close(varPipe[0 1])	Chiude una estremità della pipe.
read(varPipe[0], buffer, bufferSize)	Blocca esecuzione in attesa di dati da leggere.
strcpy(buffer, "...")	Prepara il buffer con il messaggio da inviare.
write(varPipe[1], buffer, sizeBuffer+1)	Scriva nella pipe.

Permette di comunicare fra processi correlati usando sistem-call e i descrittori di file. Il collegamento esiste fino a eliminazione esplicita o del processo.



Ogni processo ha un buffer di caratteri e una pipe (array di due celle) per scrivere e leggere con l'altro processo i dati contenuti nel buffer.

Nota: I processi sono visti come file, per questo le operazioni si chiamano come quelle dei file.

Multi-thread	Descrizione
<code>nomeFunzioneThread(arg : void*) : void*</code>	Funzione assegnata da eseguire a un thread. Necessita questa firma specifica per accettare e restituire qualsiasi tipo di dato. Serve eseguire un cast esplicito.
<code>pthread_create(&varThread, &pthreadAttribut, tFunction, &args)</code>	Crea un nuovo thread dentro al processo corrente. I parametri sono: <ol style="list-style-type: none"> 1. Variabile tipo thread. 2. Puntatore a struttura di attributi del thread. Default è NULL. 3. Funzione che sarà eseguita dal thread. 4. Puntatore a struttura contenente parametri usati dalla funzione.
<code>pthread_join(&varThread, NULL)</code>	Fa attendere processo la fine del thread indicato.

Struttura base programma multi-processo:

```

void *tFunction(void *args) {...}
int main() {
    pthread_t thread;
    ... args = ...;

    pthread_create(&thread, NULL, tFunction, &args);
}
  
```

Nota: La creazione di un processo è più lenta della creazione di un thread perchè nella prima bisogna creare un intero file descriptor, mentre nella seconda parziale.