

Shell

Shell

- la shell è un programma che si frappone fra l'utente ed il sistema operativo permettendo, mediante una interfaccia testuale, di eseguire *comandi*
- esistono molti tipi di shell diverse, ma tutte offrono funzionalità comuni:
 - esecuzione di eseguibili e cmd. interni
 - job control
 - redirezioni
 - gestione del path
 - wildcard
 - raggruppamento di cmd.
 - scripting
 - pipeline
 - variabili
 - sostituzione di cmd.
- utilizzeremo sintassi e semantica della shell più diffusa su sistemi GNU/Linux: la Bourne Again shell (bash)

Variabili

- una shell in esecuzione mantiene un insieme di variabili imperative di tipo stringa, è possibile:
 - assegnare valori a tali variabili (non è necessario dichiararle)
 - ...
 - `nome="valore della variabile"`
 - ... ed accedere al loro valore
 - `echo $nome`
- esistono due tipi di variabili: *locali* e *d'ambiente*
 - le variabili d'ambiente vengono ereditate dai processi figli
 - le nuove variabili sono locali, ma possono divenire d'ambiente
 - `export nome`

Eseguibili e comandi interni

- la shell permette di eseguire file eseguibili e comandi interni
 - file eseguibili
 - risiedono sul filesystem
 - vengono specificati con path assoluti o relativi
 - `/sbin/ifconfig`
 - `../ifconfig`
 - o cercati nei percorsi specificati nella variabile di ambiente \$PATH
 - `PATH=/usr/bin:/sbin`
 - `ifconfig`
 - viene creato un nuovo processo per ognuno di essi, la shell attende che questo termini (o che liberi la console) prima di accettare nuovi comandi

Eseguibili e comandi interni

- comandi interni
 - sono comandi implementati nella shell stessa
 - non risiedono sul filesystem
 - vengono eseguiti dallo stesso processo della shell
 - e.g. echo, cd, pwd
 - la loro doc. è accessibile con il comando interno help
- sia per eseguibili che per comandi interni, la shell interpreta la linea di comando fornita dall'utente, creando l'array dei *parametri posizionali* (associato ad ogni processo)
 - e.g.
 - ls /etc /dev par.0 = “ls”, par.1 = “/etc”, par.2 = “/dev”
 - echo pippo par.0 = “echo”, par.1 = “pippo”

Comandi interni “notevoli”

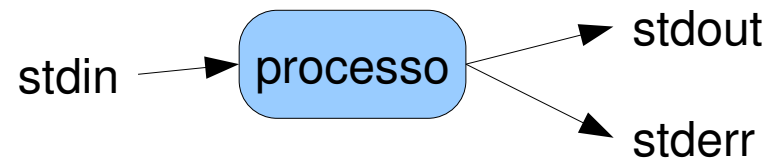
- `exec`
 - per eseguire un file eseguibile, la shell crea un nuovo processo
 - il comando `exec` permette di eseguire un file eseguibile sostituendolo al processo corrente
 - e.g. confrontate i risultati di `ls` ed `exec ls`
- `shift`
 - essendo un processo, anche la shell dispone di un array di parametri posizionali con i quali è stata invocata
 - sono accedibili grazie alle variabili `$0`, `$1`, `$2`, ...
 - `$*` è l'indice dell'ultimo parametro
 - `$@` corrisponde alla lista dei parametri
 - il comando `shift` effettua lo shift a sinistra di tutti i parametri il cui indice è ≥ 1 , `$1` viene perso

Metacaratteri

- l'interpretazione della riga di comando tratta in maniera particolare alcuni caratteri, detti *metacaratteri*:
 - > < * ? | () ; \ spazio tab cr lf
 - e.g. il carattere spazio separa gli argomenti a linea di comando per creare l'array corrispondente
- per inibire il comportamento particolare dei metacaratteri è necessario precederli con un \ (backslash)
 - questo procedimento è una delle forme possibili di *quoting*
- e.g.
 - ls foo bar (2 parametri posizionali)
 - vs
 - ls foo\ bar (1 parametro posizionale)

Redirezione

- ogni processo in esecuzione è associato ad un insieme di file aperti, 3 di essi sono predefiniti:
 - *standard input* (stdin, file descriptor 0)
 - è un file sola lettura, utilizzato per leggere input da tastiera
 - *standard output* (stdout, file descriptor 1)
 - *standard error* (stderr, file descriptor 2)
 - sono file sola scrittura, utilizzati per emettere output a video



- la shell permette di redirezionare sia i file di input che i file di output di un processo all'atto della sua esecuzione

Redirezione

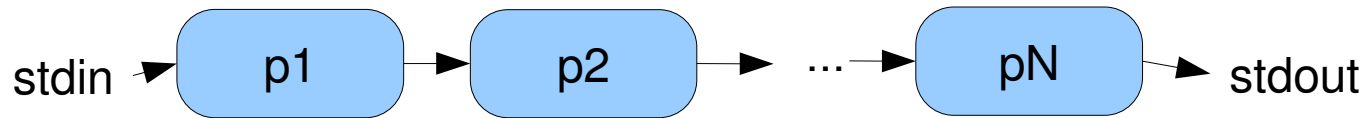
- salvataggio dell'output di processo su file
 - salvataggio di stdout: *comando > nome_file*
 - e.g. `ls /etc > etc.txt`
 - salvataggio di stderr: *comando 2> nome_file*
 - salvataggio di entrambi: *comando &> nome_file*
 - utilizzando “>>” al posto di “>” l'output del processo viene aggiunto al file destinazione anziché sostituito
 - oss: per buttare l'output di un processo possiamo redirezionarlo su `/dev/null` (e.g. `startx &> /dev/null`)
- redirezione dell'output su un altro file descriptor
 - forma generale: *comando src_fd>&dst_fd*
 - e.g. `ls asflkgjhaslkfjg 2>&1`

Redirezione

- lettura dell'input da file
 - *comando < nome_file*
 - e.g. mail zack@cs.unibo.it < testo.txt
 - e.g. sort < studenti.txt > studenti_ord.txt
- forme generali
 - apertura e ridirezione del file descriptor n, in sola scrittura
 - *comando n> target*
 - apertura e ridirezione del file descriptor n, in sola lettura
 - *comando n< target*
 - apertura e ridirezione del file descriptor n, in lettura/scrittura
 - *comando n<> target*

Pipeline

- è possibile far comunicare tra loro sequenze di processi, collegando stdout di uno di essi a stdin del successivo in strutture chiamate *pipeline*



- sintassi: *comando1 | comando2 | | comandoN*
 - e.g. `ls | wc`
 - e.g. `cat /etc/passwd | cut -f 1 -d : | sort`
 - e.g. `sort < studenti.txt | sort | uniq > studenti_ord_nodup.txt`
 - e.g. `ls -R | less`

Raggruppamento di comandi

- sequenze di comandi:
 - sintassi: *comando1 ; comando 2 ; ... ; comandoN*
 - e.g. *date; ls; pwd*
 - i comandi vengono eseguiti sequenzialmente, attendendo la terminazione
- raggruppamento di comandi:
 - sintassi: *(sequenza_di_comandi)*
 - e.g. *(date; ls; pwd)*
 - per ogni gruppo viene eseguita una nuova shell che esegue la sequenza di comandi e termina
 - date; ls; pwd > out.txt*
 - vs
 - (date; ls; pwd) > out.txt*

Sequenze condizionali

- return code
 - ogni processo, al momento della sua terminazione, restituisce un valore numerico (*return code*) intero
 - il valore 0 viene solitamente interpretato come successo, tutti gli altri come insuccesso
 - la shell ha accesso al valore di ritorno dell'ultimo processo eseguito (variabile \$?)
- sequenze condizionali
 - *comando1 && comando2*
 - esegue *comando1*, se return code = 0 esegue *comando2*
 - *comando1 || comando2*
 - esegue *comando1*, se return code ≠ 0 esegue *comando2*
 - e.g. gcc myprog.c && ./a.out

Espansione delle wildcard

- la shell permette diverse forme di *espansione* nei comandi forniti dall'utente: utilizzandole è possibile scrivere comandi abbreviati
- la forma più semplice di espansione è l'*espansione delle wildcard* che permette di definire abbreviazioni che vengono espanso con l'aiuto del filesystem
 - il carattere * viene espanso con zero o più caratteri
 - il carattere ? viene espanso con esattamente un carattere
 - stringhe che contengono questi metacaratteri vengono espanso in una lista (separata da spazi) di file esistenti sul filesystem che rispettano le regole di * e ?
 - e.g. `ls /etc/*.conf`

Espansione dei comandi

- *l'espansione dei comandi* permette di sostituire una stringa (che rappresenti un comando) con il risultato dell'esecuzione dello stesso
- sintassi
 - sintassi sh: ``comando``
 - sintassi bash: `$(comando)` più comoda per l'annidamento
- e.g.
 - `echo data di oggi `date``
 - `data=$(date)`
 - `echo nel sistema ci sono al momento `who | wc -l` utenti`
 - `risultato=`expr 1 + 2 + 3 * 7`` (notate il quoting di *)

Altre espansioni

- la sintassi che abbiamo già visto per accedere ai valori della variabili non è altro che una forma di espansione:

l'espansione delle variabili

- e.g. echo \$USER \$HOME
- la shell bash offre molti altri tipi di espansione, tra i quali:
 - espansione aritmetica e.g. \$((1+2*3))
 - brace expansion e.g. echo {a,b}{c,d}
 - tilde expansion e.g. cd ~szacchir/

Quoting

- a volte è necessario inibire una o più forme di espansione
- è possibile farlo utilizzando due nuove forme di quoting
 - single quote
 - utilizza virgolette singole `'`
 - tutto ciò che è racchiuso all'interno di `' ... '` non è suscettibile ad espansioni
 - e.g. `echo '* `whoami` $HOME'`
 - double quote
 - utilizza virgolette doppie `"`
 - tutto ciò che è racchiuso all'interno di `" ... "` è suscettibile ad espansioni di variabili e di comandi
 - e.g. `echo "* `whoami` $HOME"`

Riferimenti

- l'enciclopedia man page di bash: man bash
- Unix Power Tools, Jerry Peek et al., O'Reilly
- Unix Shell Programming, 3rd edition, Stephen Kochan et al.