Sistemi operativi: Lezione 20

Antonio Rocchia

April 27, 2022

Contents

1 Shell

La shell è un programma che permette ad un utente di interagire con il sistema operativo tramite un interfaccia testuale. Offre non solo un idle per mandare in esecuzione eseguibili ma è anche un linguaggio di scripting.

1.1 Differenti tipi di shell

Esistono molti tipi di shell:

- Bash
- C-shell
- Z-shell

La più usata(default su unix) è bash. Sono in realtà degli eseguibili sul sistema operativo. L'utente può scegliere la sua shell, la schelta viene salvata in /etc/passwd.

La shell di login è quella che richiede inizialmente i dati di accesso all'utente. Per ogni utente viene generato un processo dedicato, indipendentemente se esso sia un DE o una shell.

1.2 Ciclo di esecuzione della shell

```
Prendi da slide.
```

1.3 Accesso al sistema

Dopo l'accesso al sistema, il sistema prende da /etc/passwd le informazione dell'utente e lancia la shell scelta.

Comando passwd Il comando passwd permette di cambiare password se si conosce la precedente. L'utente admin può forzare questo cambiamento, tuttavia non può conoscere la password degli utenti.

1.4 Logout dal sistema

Per uscire da una qualsiasi shell si può usare il comando exit(). Per uscire dalla shell di login si può usare il comando logout o digitare CTRL+C

1.5 Esecuzione di un comando

Quando una shell manda in esecuzione un comando essa esegue una fork del processo shell e poi esegue un istruzione exec per mandare in esecuzione il codice dell'eseguibile richiesto.

Grazie a questo meccanismo si può usare la shell come interprete di programmi definiti dall'utente.

Solitamente i comandi vengono lanciati in *foreground* (la shell si mette in attesa, aspettanto che il comando finisca la sua esecuzione), tuttavia è possibile chiedere alla shell di eseguire i comandi in *background*

,

```
loop forever
<login>
do{
     scanf(comando)
     pid=fork()
     if(pid==0)
      execlp(comando)
}
```

1.6 Comandi input/output

Esempi di comandi UNIX:

- grep, ricerca di testo
- tee, scrive l'input sia su file che su output
- sort, Ordina alfabeticamente le righe
- rev, inverte l'ordine delle linee di file
- cut, Selezione colonne da file
- awk, ricerca di testo ed esecuzione di comandi `awk '/^In/ { print }'file1`
- expr, valutazione di espressioni

1.7 Ridirezione di comandi

Possiamo ridirezionare l'input/output di comandi su file. In bash abbiamo tre operatori di ridirezionamento:

- '<', ridireziono in lettura -> "comando < file_input"
- '>', ridireziono in scrittura(tronco) -> "comando > file_output"
- '»', ridireziono in scrittura(append) -> "comando > file_output"

```
ls -l > file
sort < file > file2
echo ciao >> file
```

Le stringhe possono essere delimitate opzionalmente con i doppi apici ("). `> file1` crea un file chiamato file1

1.8 Piping

L'output di un programma può essere ridirezionato verso l'input di un altro. In DOS viene creato un file temporaneo dove viene salvato l'output del primo comando che viene usato come input per il secondo. In unix viene usata una pipe.

L'operatore di piping è '|'.

Conta gli utenti collegati

```
who | wc -l
```

Un esempio di piping complesso

```
ls -l | grep ^d | rev | cut -d' ' -f1 | rev
```

Questo comando lista i file del direttorio corrente, tale output viene filtrato da grep che prende solo le righe che iniziano con 'd' ovvero le directories. Invertiamo l'ordine della lista. Usiamo cut per prendere il primo campo che in questo caso è l'ultimo campo del comando ls che è il nome del file, poiche abbiamo usato rev dobbiamo reinvertire il nome del file per renderlo leggibile.

1.9 Metacaratteri

La shell riconoscere dei caratteri speciali (wild card)

- '*' viene interpretato come una qualunque stringa di 0 o più caratteri.
- '?' un qualunque carattere singolo
- '[zfc]' un qualunque carattere compreso tra quello nell'insieme.
- '[a-d]' un qualunque carattere compreso nell'intervallo
- '#' un commanto fino alla fine della linea
- 'Éscape, segnala di non interpretare il carettere successivo come carattere speciale.

`ls ese*` elenca tutti i file che iniziano con la stringa ese. `ls [a-p,1-7]*[c,f,d]?` elenca i file i cui nomi hanno come iniziale un carattere compreso tra a e p oppure 1 e 7 e il cui penultimo carattere sia c, f o d. `ls *\**` elenca i file che contegono nel nome, in qualsiasi posizione il carattere *

1.10 Variabili nella shell

In ogni shell è possibile definire delle variabili che possono essere richiamate dai programmi successivi.

La shell fa una distinzione di semantica nelle variabili. Quando definiamo il nome della variabile scriviamo VAR=<valore>. Quando recuperiamo il valore di var scriviamo $VAR. `echo $VAR` da come output valore mentre `echo VAR` da come output VAR

Un assegnamento tra variabili è dichiarato come `VAR=$OTHER_VAR`, assegnamo a VAR il valore di OTHER_VAR.

Appendere una directory corrente alla PATH

```
PATH=$PATH:/usr/local/bin
```

In questo esempio assegno alla variabile PATH il valore di PATH a cui aggiungo 'usr/local/bin' alla path.

1.11 Ambiente di esecuzione

Ogni shell esegue in un ambiente in cui possiamo definire variabili, le *variabili di ambiente*. Ogni shell condivide il proprio ambiente di esecuzione con i figli che genera.

`set` stampa tutte le variabili d'ambiente definite al momento in cuila shell esegue il comando.

1.12 Espressioni

Nella shell un blocco delimitato tra due ‘ ‘ (backtick) viene interpretato come un comando. Il backtick dice alla shell di valutare l'espressione all'interno del blocco e di sostituire il valore valutato come stringa nel punto in cui è presente il blocco.

```
echo 3 + 1 = ‘expr 3 + 1‘
echo 3 + 1 = expr 3 + 1
```

da come output

```
3 + 1 = 4
3 + 1 = expr 3 + 1
```

2 Shell scripting

Si possono creare file che contengono comandi da eseguire in sequenza.

2.1 Espansione

Quando mettiamo in esecuzione i comandi su una shell, prima di eseguirlo la shell prepara e collega i comandi.

Sequenza dei passi di sostituzione:

1. Sostituzione dei comandi tra backquote
2. Sostituzione delle variabili e parametri
3. Sostituzione dei metacaratteri nelle stringhe.

In alcuni casi è necessario privare i caratteri del loro significato

- \ protegge dall'espansione il carattere successivo
- ” ” protegge dall'espansione qualsiasi cosa ci sia all'interno
- "" proteggono dalle espansioni qualsiasi cosa ci sia all'interno ad eccezione di $, ‘ e

Ad esempio `echo "<‘pwd‘>"` mi restituisce <home/rokomia> mentre `echo '<‘pwd‘>'` mi restituisce <‘pwd‘>

2.2 File comandi

Un file comandi è composto da una sequenza di comandi (uno per riga) e può contenere:

- Statement per il controllo di flusso
- variabili
- passaggio di argomenti

Gli statement disponibili dipendono dalla shell che si utilizza. I file comandi vengono interpretati e non compilati. Ogni file comandi deve essere eseguibile (chmod +x).