Fondamenti di Informatica (Elettronici)

Intro alla shell – Lezione 27

7 dicembre 2020

Bash shell [1]

La shell è il programma più importante in un sistema operativo, dopo il kernel. È in pratica il mezzo con cui si comunica con il sistema e attraverso il quale si avviano e si controlla l'esecuzione degli altri programmi.

1. 43. Introduzione alla shell
2. 44. Bash: avvio e conclusione
3. Comandi di shell
4. Bash cheat sheet

[1] Tratto da: Daniele Giacomini, Appunti Linux – appunti per un desiderio di informatica libera, disponibile sotto Licenza 'GNU GENERAL PUBLIC LICENSE - Version 2' (in Appendice).

Section 1

43. Introduzione alla shell

Invito (prompt) della shell

Quando una shell attende ed esegue i comandi impartiti dall'utente, si trova in una modalità di funzionamento interattivo.

La disponibilità da parte della shell di ricevere comandi viene evidenziata dall'apparizione sullo schermo del terminale di un messaggio di invito o prompt.

Questo, per lo più, è composto da simboli e informazioni utili all'utente per tenere d'occhio il contesto in cui sta operando.

In questo senso, l'invito è un elemento importante della shell, e ancora più importante è la possibilità di configurarlo in base alle proprie esigenze.

Il concetto di «invito» riguarda tutti i programmi che richiedono un'interazione con l'utente attraverso una riga di comando.

Storico (history) dei comandi

Lo storico dei comandi (o «storia», se si preferisce il termine) è un registro degli ultimi comandi inseriti dall'utente.

Quando la shell lo gestisce, l'utente è in grado di ripescare facilmente, ed eventualmente modificare, un comando utilizzato poco prima senza doverlo riscrivere completamente.

Comandi interni

La maggior parte delle shell mette a disposizione una serie di comandi interni (o comandi incorporati) che vengono richiamati nello stesso modo con cui si avvia un programma normale.

Solitamente, se esiste un programma con lo stesso nome di un comando, è quest'ultimo, cioè il comando, ad avere la precedenza.

Di solito, i programmi standard che hanno lo stesso nome di comandi interni delle shell principali, svolgono un compito simile.

Spesso, questo fatto è causa di equivoci fastidiosi: alle volte non si è in grado di capire il motivo per il quale un certo programma non funziona esattamente come ci si aspetterebbe.

Alias

Alcune shell permettono la definizione di nuovi comandi in forma di alias di comandi già esistenti.

L'utilità di questo sta nella possibilità di permettere l'uso di nomi differenti per uno stesso risultato, oppure per definire l'utilizzo sistematico di opzioni determinate.

Per comprendere il senso di questo si può considerare un esempio.

Esempio

Creiamo l'alias `dir` che in realtà esegue il comando `ls -l`.

```
$ alias <newcommand>='<old command>'
```

Ambiente

Ogni programma in funzione nel sistema ha un proprio ambiente definito in base a delle variabili di ambiente.

Le variabili di ambiente sono un mezzo elementare e pratico di configurazione del sistema: i programmi, a seconda dei loro compiti e del loro contesto, cercano di leggere alcune variabili di loro interesse, e in base al contenuto di queste adeguano il loro comportamento.

L'ambiente consegnato a ogni programma che viene messo in esecuzione, è controllato dalla shell che è in grado di assegnare ambienti diversi a programmi diversi.

La shell può quindi creare, modificare e leggere queste variabili, cosa particolarmente utile per la realizzazione di `file script`.

Pipeline

La shell mette in esecuzione i comandi ed è in grado di ridirigere il flusso di dati standard: standard input, standard output e standard error.

Questa caratteristica è importantissima per la realizzazione di comandi complessi attraverso l'elaborazione successiva da parte di una serie di programmi.

Di fatto, ogni comando, anche se composto dalla richiesta di esecuzione di un solo programma, è una pipeline dal punto di vista della shell.

Script

Con il termine `script` si identifica un programma scritto ed eseguito nella sua forma sorgente senza l'intervento di alcuna compilazione.

Normalmente, le `shell` sono in grado di eseguire dei `file script`, scritti secondo il linguaggio loro adatto.

Per convenzione, gli script di shell e anche di altri linguaggi interpretati, iniziano con una riga che specifica il programma in grado di interpretarli.

#!/bin/sh

Questa riga, per esempio, è l'inizio di uno script che deve essere interpretato dal programma `/bin/sh`, ovvero dalla shell Bourne o altra compatibile.

```
#!/bin/bash
```

#!/bin/sh

Questa riga, per esempio, è l'inizio di uno script che deve essere interpretato dal programma `/bin/sh`, ovvero dalla shell Bourne o altra compatibile.

```
#!/bin/bash
```

1. Si chiama shebang o linea "bang".

#!/bin/sh

Questa riga, per esempio, è l'inizio di uno script che deve essere interpretato dal programma `/bin/sh`, ovvero dalla shell Bourne o altra compatibile.

```
#!/bin/bash
```

1. Si chiama `shebang` o linea "`bang`".
2. Non è altro che il percorso assoluto per l'interprete `bash`.

#!/bin/sh

Questa riga, per esempio, è l'inizio di uno script che deve essere interpretato dal programma `/bin/sh`, ovvero dalla shell Bourne o altra compatibile.

```
#!/bin/bash
```

1. Si chiama `shebang` o linea "`bang`".
2. Non è altro che il percorso assoluto per l'interprete `bash`.
3. Consiste in un carattere `number` (#) e un punto esclamativo (!), seguiti dal percorso completo dell'interprete come `/bin/bash`.

#!/bin/sh

Questa riga, per esempio, è l'inizio di uno script che deve essere interpretato dal programma `/bin/sh`, ovvero dalla shell Bourne o altra compatibile.

```
#!/bin/bash
```

1. Si chiama `shebang` o linea "`bang`".
2. Non è altro che il percorso assoluto per l'interprete `bash`.
3. Consiste in un carattere `number` (#) e un punto esclamativo (!), seguiti dal percorso completo dell'interprete come `/bin/bash`.
4. Tutti gli script vengono eseguiti utilizzando l'interprete specificato sulla prima riga.

#!/bin/sh

Questa riga, per esempio, è l'inizio di uno script che deve essere interpretato dal programma `/bin/sh`, ovvero dalla shell Bourne o altra compatibile.

```
#!/bin/bash
```

1. Si chiama `shebang` o linea "`bang`".
2. Non è altro che il percorso assoluto per l'interprete `bash`.
3. Consiste in un carattere `number` (#) e un punto esclamativo (!), seguiti dal percorso completo dell'interprete come `/bin/bash`.
4. Tutti gli script vengono eseguiti utilizzando l'interprete specificato sulla prima riga.
5. Quasi tutti gli script `bash` spesso iniziano con `#!/bin/bash` (assumendo che `bash` sia stato installato in `/bin`)

#!/bin/sh

Questa riga, per esempio, è l'inizio di uno script che deve essere interpretato dal programma `/bin/sh`, ovvero dalla shell Bourne o altra compatibile.

```
#!/bin/bash
```

1. Si chiama `shebang` o linea "`bang`".
2. Non è altro che il percorso assoluto per l'interprete `bash`.
3. Consiste in un carattere `number` (#) e un punto esclamativo (!), seguiti dal percorso completo dell'interprete come `/bin/bash`.
4. Tutti gli script vengono eseguiti utilizzando l'interprete specificato sulla prima riga.
5. Quasi tutti gli script `bash` spesso iniziano con `#!/bin/bash` (assumendo che `bash` sia stato installato in `/bin`)
6. Ciò garantisce che `bash` verrà utilizzato per interpretare lo script, anche se viene eseguito in un'altra shell.

#!/bin/sh

Questa riga, per esempio, è l'inizio di uno script che deve essere interpretato dal programma `/bin/sh`, ovvero dalla shell Bourne o altra compatibile.

```
#!/bin/bash
```

1. Si chiama `shebang` o linea "`bang`".
2. Non è altro che il percorso assoluto per l'interprete `bash`.
3. Consiste in un carattere `number` (#) e un punto esclamativo (!), seguiti dal percorso completo dell'interprete come `/bin/bash`.
4. Tutti gli script vengono eseguiti utilizzando l'interprete specificato sulla prima riga.
5. Quasi tutti gli script `bash` spesso iniziano con `#!/bin/bash` (assumendo che `bash` sia stato installato in `/bin`)
6. Ciò garantisce che `bash` verrà utilizzato per interpretare lo script, anche se viene eseguito in un'altra shell.
7. Lo `shebang` è stato introdotto da Dennis Ritchie tra la versione 7 di Unix e la versione 8 dei Bell Labs. È stato poi aggiunto anche alla versione BSD a Berkeley.

#!/bin/sh

Questa riga, per esempio, è l'inizio di uno script che deve essere interpretato dal programma `/bin/sh`, ovvero dalla shell Bourne o altra compatibile.

```
#!/bin/bash
```

1. Si chiama `shebang` o linea "`bang`".
2. Non è altro che il percorso assoluto per l'interprete `bash`.
3. Consiste in un carattere `number` (#) e un punto esclamativo (!), seguiti dal percorso completo dell'interprete come `/bin/bash`.
4. Tutti gli script vengono eseguiti utilizzando l'interprete specificato sulla prima riga.
5. Quasi tutti gli script `bash` spesso iniziano con `#!/bin/bash` (assumendo che `bash` sia stato installato in `/bin`)
6. Ciò garantisce che `bash` verrà utilizzato per interpretare lo script, anche se viene eseguito in un'altra shell.
7. Lo `shebang` è stato introdotto da Dennis Ritchie tra la versione 7 di Unix e la versione 8 dei Bell Labs. È stato poi aggiunto anche alla versione BSD a Berkeley.

#!/bin/sh

Questa riga, per esempio, è l'inizio di uno script che deve essere interpretato dal programma `/bin/sh`, ovvero dalla shell Bourne o altra compatibile.

```
#!/bin/bash
```

1. Si chiama `shebang` o linea "`bang`".
2. Non è altro che il percorso assoluto per l'interprete `bash`.
3. Consiste in un carattere `number` (#) e un punto esclamativo (!), seguiti dal percorso completo dell'interprete come `/bin/bash`.
4. Tutti gli script vengono eseguiti utilizzando l'interprete specificato sulla prima riga.
5. Quasi tutti gli script `bash` spesso iniziano con `#!/bin/bash` (assumendo che `bash` sia stato installato in `/bin`)
6. Ciò garantisce che `bash` verrà utilizzato per interpretare lo script, anche se viene eseguito in un'altra shell.
7. Lo `shebang` è stato introdotto da Dennis Ritchie tra la versione 7 di Unix e la versione 8 dei Bell Labs. È stato poi aggiunto anche alla versione BSD a Berkeley.

Caratteri jolly o metacaratteri

Una caratteristica delle shell è la possibilità di effettuare sostituzioni, o espansioni, nel comando impartito interattivamente o contenuto in un programma script.

I caratteri jolly, o metacaratteri, sono quei simboli utilizzati per fare riferimento facilmente a gruppi di file o directory.

Nei sistemi Unix sono le shell a occuparsi della traduzione dei caratteri jolly.

In questo modo, una riga di comando che ne contiene, viene trasformata dalla shell che fornisce così, al programma da avviare, l'elenco completo di file e directory che si ottengono dall'espansione di questi caratteri speciali.

È utile ricordare che alcuni testi fanno riferimento a questo concetto con il termine `globbing`.

Variabili e parametri

Come accennato, le shell permettono di creare o modificare il contenuto di variabili di ambiente.

Queste variabili possono essere utilizzate per la costruzione di comandi, ottenendo così la sostituzione con il valore che contengono, prima dell'esecuzione di questi.

Nello stesso modo, i parametri, che sono un tipo particolare di variabili a sola lettura, possono essere usati nelle righe di comando.

Di solito si tratta degli argomenti passati a uno script.

Section 2

44. Bash: avvio e conclusione

Avvio di Bash

L'eseguibile della shell Bash è `bash`, che si trova normalmente nella directory `/bin/`.

```
$ bash [<opzioni>] [<file-script>] [<argomenti>]
```

Si distinguono fondamentalmente due tipi di modalità di funzionamento della shell Bash: interattiva e non interattiva.

Quando l'eseguibile `bash` viene avviato con l'indicazione del nome di un file, questo tenta di eseguirlo come uno script (in tal caso non conta che il file abbia i permessi di esecuzione e nemmeno che contenga la dichiarazione iniziale `#!/bin/bash`).

Gli eventuali argomenti che possono seguire il nome del file, vengono passati allo script in forma di parametri (come viene descritto più avanti).

La shell Bash è potenzialmente compatibile con diversi altri tipi di shell.

File di configurazione della shell

La tabella: [configurazione bash] mostra, in particolare, la sequenza di file di configurazione utilizzati a seconda del tipo di emulazione preferito.

Shell interattiva

La shell è interattiva quando interagisce con l'utente e di conseguenza mostra un invito a inserire dei comandi.

L'eseguibile `bash` può essere avviato eventualmente in modo esplicitamente interattivo utilizzando l'opzione `-i`.

Quando la shell Bash funziona in modo interattivo, se necessario, crea la variabile di ambiente `PS1` che serve a contenere l'invito, e il parametro `$-` contiene anche la lettera `i`.

Una shell interattiva può a sua volta essere una "shell di login" o meno.

La distinzione serve alla shell per determinare quali file di configurazione utilizzare.

Shell non interattiva

Una shell non interattiva è di norma dedicata a eseguire uno script.

Nel momento dell'avvio in questa modalità, la shell Bash controlla il contenuto della variabile di ambiente `BASH_ENV`: se questa variabile non è vuota esegue il file nominato al suo interno.

In pratica, attraverso questa variabile si indica un file di configurazione che si vuole sia eseguito dalla shell prima dello script.

In situazioni normali questa variabile è vuota, oppure non esistente del tutto.

Interpretazione degli argomenti successivi

Se restano degli argomenti dopo le opzioni e non è stato usato `-c` o `-s`, il primo di questi argomenti viene interpretato come il nome di un file di comandi di shell: uno script di shell.

Se la shell viene avviata in questo modo, viene assegnato al parametro `$0` il nome di questo script e ai parametri posizionali (`$1`, `$2`, `$3`, ecc.) il resto degli argomenti.

La shell legge ed esegue i comandi di questo script e quindi termina l'esecuzione.

Il valore restituito alla fine della sua esecuzione è quello dell'ultimo comando eseguito dallo script.

Conclusione

La conclusione del funzionamento della shell, quando si trova in modalità interattiva, si ottiene normalmente attraverso il comando interno `exit`, oppure eventualmente con il comando interno `logout` se si tratta di una shell di login.

Se invece si tratta di una shell avviata per interpretare uno script, questa termina silenziosamente alla conclusione dello script stesso.

Conclusione

La conclusione del funzionamento della shell, quando si trova in modalità interattiva, si ottiene normalmente attraverso il comando interno `exit`, oppure eventualmente con il comando interno `logout` se si tratta di una shell di login.

Se invece si tratta di una shell avviata per interpretare uno script, questa termina silenziosamente alla conclusione dello script stesso.

Tutti i comandi eseguiti senza errori terminano silenziosamente

Section 3

Comandi di shell

1/3

La riga di comando è qualcosa che ogni sviluppatore dovrebbe imparare e implementare nella propria routine quotidiana

È diventato un coltellino svizzero con funzioni dietro comandi apparentemente semplici, che ti consentono di ottenere un maggiore controllo del tuo sistema, diventare più produttivo e molto altro

Ad esempio, puoi scrivere script per automatizzare attività quotidiane che richiedono molto tempo e persino eseguire rapidamente il commit e il push del codice in un repository Git con pochi semplici comandi.

2/3

In questa lezione vedremo la shell Bash (Bourne Again SHell), che è un'interfaccia a riga di comando (CLI) ed è attualmente la shell più utilizzata

Questa è una breve introduzione ai comandi più popolari, quando è più probabile che li userai e come estenderli con le opzioni

Più avanti in questa lezione imparerai come creare i tuoi comandi personalizzati (alias), permettendoti di creare scorciatoie per un singolo comando o un gruppo di comandi.

3/3

In fin dei conti, se non conosci la riga di comando, non stai utilizzando il tuo computer al massimo delle sue potenzialità.

- Alcuni comandi possono essere utilizzati senza opzioni o specificando file.

3/3

In fin dei conti, se non conosci la riga di comando, non stai utilizzando il tuo computer al massimo delle sue potenzialità.

- Alcuni comandi possono essere utilizzati senza opzioni o specificando file.
- Scopri come automatizzare gli script.

3/3

In fin dei conti, se non conosci la riga di comando, non stai utilizzando il tuo computer al massimo delle sue potenzialità.

- Alcuni comandi possono essere utilizzati senza opzioni o specificando file.
- Scopri come automatizzare gli script.
- Nota rapida: tutto ciò che è racchiuso in parentesi quadre [] significa che è facoltativo.

3/3

In fin dei conti, se non conosci la riga di comando, non stai utilizzando il tuo computer al massimo delle sue potenzialità.

- Alcuni comandi possono essere utilizzati senza opzioni o specificando file.
- Scopri come automatizzare gli script.
- Nota rapida: tutto ciò che è racchiuso in parentesi quadre [] significa che è facoltativo.
- per approfondimenti: Sintesi dei comandi principali

Section 4

Bash cheat sheet

— Elenca il contenuto della directory

`ls` è probabilmente il comando più comune.

Molte volte lavorerai in una directory e avrai bisogno di sapere quali file si trovano lì.

Il comando ls consente di visualizzare rapidamente tutti i file all'interno della directory specificata.

Sintassi: `ls [option(s)] [file(s)]`

Opzioni comuni: `-a, -l`

- Stampa il testo nella finestra del terminale

`echo` stampa il testo nella finestra del terminale e viene tipicamente utilizzato negli script della shell e nei file batch per visualizzare il testo di stato sullo schermo o su un file del computer.

Echo è anche particolarmente utile per mostrare i valori delle variabili ambientali, che dicono alla shell come comportarsi mentre un utente lavora dalla riga di comando o negli script.

Sintassi: `echo [option(s)] [string(s)]`

Opzioni comuni: `-e, -n`

- Crea un file

`touch` sarà il modo più semplice per creare nuovi file, ma può anche essere utilizzato per modificare i timestamp su file e / o directory.

Puoi creare tutti i file che desideri con un singolo comando senza preoccuparti di sovrascrivere i file con lo stesso nome.

Sintassi: `touch [option(s)] file_name(s)`

Opzioni comuni: `-a, -m, -r, -d`

- Crea (make) una directory

`mkdir` è un utile comando che puoi usare per creare directory.

È possibile creare contemporaneamente un numero qualsiasi di directory, il che può accelerare notevolmente il processo.

Sintassi: `mkdir [option(s)] directory_name(s)`

Opzioni comuni: `-m, -p, -v`

- ricerca

`grep` viene utilizzato per cercare nel testo o nel sistema i pattern specificati dall'utente.

È uno dei comandi più utili e potenti.

Ci sono spesso scenari in cui ti verrà chiesto di trovare una particolare stringa o pattern all'interno di un file, ma non sai da dove iniziare a cercare, è qui the grep è estremamente utile.

Sintassi: `grep [option(s)] pattern [file(s)]`

Opzioni comuni: `-i, -c, -n`

- Stampa il manuale o ottieni aiuto per un comando

Il comando `man` è il tuo manuale ed è molto utile quando hai bisogno di capire cosa fa un comando.

Ad esempio, se non sapessi cosa fa il comando `rmdir`, potresti usare il comando `man` per scoprirlo.

Sintassi: `man [option(s)] keyword(s)`

Opzioni comuni: `-w`, `-f`, `-b`

- Stampa la directory di lavoro

`pwd` viene utilizzato per stampare la directory corrente in cui ti trovi.

Ad esempio, se hai più terminali attivi e hai bisogno di ricordare la directory esatta in cui stai lavorando, allora `pwd` te lo dirà.

Sintassi: `pwd [option(s)]`

Opzioni comuni: `options aren't typically used with pwd`

- Cambia directory

`cd` cambierà la directory in cui ti trovi in modo che tu possa ottenere informazioni, manipolare, leggere, ecc. i diversi file e directory nel sistema.

Sintassi: `cd [option(s)] directory`

Opzioni comuni: `options aren't typically used with cd`

- Sposta o rinomina la directory

`mv` viene utilizzato per spostare o rinominare le directory.

Senza questo comando, dovresti rinominare individualmente ogni file che è noioso.

`mv` ti consente di rinominare i file batch che possono farti risparmiare un sacco di tempo.

Sintassi: `mv [option(s)] argument(s)`

Opzioni comuni: `-i, -b`

- Rimuovi directory

`rmdir` rimuoverà le directory vuote.

Questo può aiutarti a liberare spazio sul tuo computer e mantenere organizzati file e cartelle.

È importante notare che esistono due modi per rimuovere le directory: rm e rmdir.

La distinzione tra i due è che `rmdir` eliminerà solo directory vuote, mentre `rm` rimuoverà directory e file indipendentemente dal fatto che contengano dati o meno.

Sintassi: `rmdir [option(s)] directory_names`

Opzioni comuni: `-p`

- Individua un file o una directory specifica

`Locate` è di gran lunga il modo più semplice per trovare un file o una directory.

Puoi mantenere la tua ricerca ampia se non sai esattamente cosa stai cercando oppure puoi restringere l'ambito utilizzando caratteri jolly o espressioni regolari.

Sintassi: `locate [option(s)] file_name(s)`

Opzioni comuni: `-q, -n, -i`

- visualizza il contenuto di un file di testo

Il comando `less` ti consente di visualizzare i file senza aprire un editor.

È più veloce da usare e non c'è possibilità che tu modifichi inavvertitamente il file.

Sintassi: `less file_name`

Opzioni comuni: `-e, -f, -n`

- Mostra tutti i comandi, alias e funzioni disponibili

`compgen` è un comando utile quando è necessario fare riferimento a tutti i comandi, alias e funzioni disponibili.

`compgen`: An Awesome Command To List All Linux Commands

Sintassi: `compgen [option(s)]`

Opzioni comuni: `-a`, `-c`, `-d`

> - reindirizza lo stdout

Il carattere > è l'operatore di reindirizzamento.

> prende l'*output del___ comando precedente___* che normalmente vedresti nel terminale e lo invia a un file che gli dai, oppure a un altro comando.

Ad esempio, `$ echo "contenuto di file1" > file1`

Qui crea un file chiamato `file1` e vi inserisce la stringa.

Sintassi: >

Opzioni comuni: `n/a`

- Legge un file, crea un file e concatena i file

`cat` è uno dei comandi più versatili e svolge tre funzioni principali sui file: visualizzarli, combinarne le copie e crearne di nuovi.

Sintassi: `cat [option(s)] [file_name(s)] [-] [file_name(s)]`

Opzioni comuni: `-n`

— Pipe

Una pipe accetta l'output standard di un comando e lo passa come input a un altro.

Esempio: `cat /Users/paoluzzi/.julia/packages/ArrayLayouts/L5sKF/src/factori | less`

Sintassi: `|`

Opzioni comuni: `n/a`