Comandi avanzati per gli utenti Linux

I comandi avanzati offrono una maggiore personalizzazione e controlli in situazioni più specialistiche una volta acquisita familiarità con i comandi di base.

Obiettivi : In questo capitolo, i futuri amministratori Linux impareranno:

✔ alcuni comandi utili non trattati nel capitolo precedente.
✔ alcuni comandi avanzati.

🏁 **comandi utente**, **Linux**

Conoscenza: ⭐
Complessità: ⭐⭐⭐

Tempo di lettura: 20 minuti

---

comando `uniq`

Il comando `uniq` è un comando molto potente, usato con il comando `sort` , soprattutto per l'analisi dei file di registro. Ti consente di ordinare e visualizzare le voci rimuovendo i duplicati.

Per illustrare come funziona il comando `uniq` , usiamo un file `firstnames.txt` contenente un elenco di nomi primi:

```
antoine
xavier
steven
patrick
xavier
antoine
antoine
steven
```

> **Nota**
>
> `uniq` richiede che il file di input sia ordinato perché confronta solo le righe consecutive.

Senza argomenti, il comando `uniq` non visualizza le righe identiche che si susseguono nel file `firstnames.txt`:

```
$ sort firstnames.txt | uniq
antoine
patrick
steven
xavier
```

Per visualizzare solo le righe che appaiono solo una volta, utilizzare l'opzione `-u`:

```
$ sort firstnames.txt | uniq -u
patrick
```

Al contrario, per visualizzare solo le righe che compaiono almeno due volte nel file, utilizzare l'opzione `-d`:

```
$ sort firstnames.txt | uniq -d
antoine
steven
xavier
```

Per eliminare semplicemente linee che appaiono solo una volta, utilizzare l'opzione `-D`:

```
$ sort firstnames.txt | uniq -D
antoine
antoine
antoine
steven
steven
xavier
xavier
```

Infine, contare il numero di occorrenze di ciascuna linea, utilizzare l'opzione `-c`:

```
$ sort firstnames.txt | uniq -c
      3 antoine
      1 patrick
      2 steven
      2 xavier
```

```
$ sort firstnames.txt | uniq -cd
      3 antoine
      2 steven
      2 xavier
```

comando `xargs`

Il comando `xargs` consente la costruzione e l'esecuzione delle linee di comando da input standard.

Il comando `xargs` legge lo spazio bianco o gli argomenti delimitati da linefeed dall'ingresso standard, ed esegue il comando (`/bin/echo` per impostazione predefinita.) una o più volte utilizzando gli argomenti iniziali seguiti dagli argomenti letti dall'ingresso standard.

Un primo e più semplice esempio sarebbe il seguente:

```
$ xargs
use
of
xargs
<CTRL+D>
use of xargs
```

Il comando `xargs` attende un input dallo standard input **stdin**. Sono state inserite tre linee. La fine dell'ingresso dell'utente in `xargs` è specificato dalla sequenza di tasti `CTRL`+`D`. `xargs` esegue quindi il comando predefinito `echo` seguito dai tre argomenti corrispondenti all'input dell'utente, vale a dire:

```
$ echo "use" "of" "xargs"
use of xargs
```

È possibile specificare un comando da far eseguire a `xargs`.

Nell'esempio seguente, `xargs` eseguirà il comando `ls -ld` sul set di cartelle specificate nell'input standard:

```
$ xargs ls -ld
/home
/tmp
/root
<CTRL+D>
drwxr-xr-x. 9 root root 4096  5 avril 11:10 /home
dr-xr-x---. 2 root root 4096  5 avril 15:52 /root
drwxrwxrwt. 3 root root 4096  6 avril 10:25 /tmp
```

In pratica, il comando `xargs` esegue il comando `ls -ld /home /tmp /root`.

Cosa succede se il comando da eseguire non accetta argomenti multipli, come nel caso del comando `find`?

```
$ xargs find /var/log -name
*.old
*.log
find: paths must precede expression: *.log
```

Il comando `xargs` tenta di eseguire il comando `find` con più argomenti dietro l'opzione

`-name`, questo causa la generazione di un errore in `find`:

```
$ find /var/log -name "*.old" "*.log"
find: paths must precede expression: *.log
```

In questo caso, il comando `xargs` deve essere costretto ad eseguire il comando `find` più volte (una volta per riga immessa come ingresso standard). L'opzione `-L` Seguito da un **intero** consente di specificare il numero massimo di voci da elaborare con il comando contemporaneamente:

```
$ xargs -L 1 find /var/log -name
*.old
/var/log/dmesg.old
*.log
/var/log/boot.log
/var/log/anaconda.yum.log
/var/log/anaconda.storage.log
/var/log/anaconda.log
/var/log/yum.log
/var/log/audit/audit.log
/var/log/anaconda.ifcfg.log
/var/log/dracut.log
/var/log/anaconda.program.log
<CTRL+D>
```

Per specificare entrambi gli argomenti sulla stessa riga, utilizzare l'opzione `-n 1`:

```
$ xargs -n 1 find /var/log -name
*.old *.log
/var/log/dmesg.old
/var/log/boot.log
/var/log/anaconda.yum.log
/var/log/anaconda.storage.log
/var/log/anaconda.log
/var/log/yum.log
/var/log/audit/audit.log
/var/log/anaconda.ifcfg.log
/var/log/dracut.log
/var/log/anaconda.program.log
<CTRL+D>
```

Caso di esempio di un backup con un `tar` basato su una ricerca:

```
$ find /var/log/ -name "*.log" -mtime -1 | xargs tar cvfP /root/log.tar
$ tar tvfP /root/log.tar
-rw-r--r-- root/root      1720 2017-04-05 15:43 /var/log/boot.log
-rw-r--r-- root/root    499270 2017-04-06 11:01 /var/log/audit/audit.log
```

La caratteristica speciale del comando `xargs` è che posiziona l'argomento di input alla fine del comando chiamato. Questo funziona molto bene con l'esempio sopra riportato dal momento che i file passati formano l'elenco dei file da aggiungere all'archivio.

Utilizzando l'esempio del comando `cp`, per copiare un elenco di file in una directory, questo elenco di file verrà aggiunto alla fine del comando... ma ciò che il comando `cp` si aspetta alla fine del comando è la destinazione. Per farlo, si può usare l'opzione `-I` per inserire gli argomenti di input in un punto diverso dalla fine della riga.

```
$ find /var/log -type f -name "*.log" | xargs -I % cp % /root/backup
```

L'opzione `-I` consente di specificare un carattere (il carattere `%` nell'esempio precedente) in cui verranno inseriti i file di input di `xargs`.

pacchetto `yum-utils`

Il pacchetto `yum-utils` è una raccolta di utilità, realizzate per `yum` da vari autori, che ne rendono più facile e potente l'uso.

> **Nota**
>
> Mentre `yum` è stato sostituito da `dnf` in Rocky Linux 8, il nome del pacchetto è rimasto `yum-utils`, sebbene possa essere installato anche come `dnf-utils`. Queste sono le classiche utilities YUM implementate come shims CLI sopra a DNF per mantenere la retrocompatibilità con `yum-3`.

Ecco alcuni esempi di utilizzo:

- comando `repoquery`

Il comando `repoquery` viene utilizzato per interrogare i pacchetti nel repository.

Esempi di utilizzo:

- Visualizza le dipendenze di un pacchetto (può essere un pacchetto software che è stato installato o non è stato installato), equivalente a `dnf deplist <nome-pacchetto>`

  ```
  repoquery --requires <package-name>
  ```

- Visualizza i file forniti da un pacchetto installato (non funziona per i pacchetti che non sono installati), Equivalente a `rpm -ql <package-name>`

  ```
  $ repoquery -l yum-utils
  /etc/bash_completion.d
  /etc/bash_completion.d/yum-utils.bash
  /usr/bin/debuginfo-install
  ```

```
/usr/bin/find-repos-of-install
/usr/bin/needs-restarting
/usr/bin/package-cleanup
/usr/bin/repo-graph
/usr/bin/repo-rss
/usr/bin/repoclosure
/usr/bin/repodiff
/usr/bin/repomanage
/usr/bin/repoquery
/usr/bin/reposync
/usr/bin/repotrack
/usr/bin/show-changed-rco
/usr/bin/show-installed
/usr/bin/verifytree
/usr/bin/yum-builddep
/usr/bin/yum-config-manager
/usr/bin/yum-debug-dump
/usr/bin/yum-debug-restore
/usr/bin/yum-groups-manager
/usr/bin/yumdownloader
…
```

- comando `yumdownloader`:

Il comando `yumdownloader` scarica i pacchetti RPM dai repository. Equivalente a `dnf scaricare --downloadonly --downloaddir ./ package-name`

> **Nota**
>
> Questo comando è molto utile per creare rapidamente un repository locale di alcuni rpm!

Esempio: `yumdownloader` scaricherà il pacchetto rpm ***repoquery*** e tutte le sue dipendenze:

```
$ yumdownloader --destdir /var/tmp --resolve samba
o
$ dnf download --downloadonly --downloaddir /var/tmp  --resolve  samba
```

Opzioni	Commenti
`--destdir`	I pacchetti scaricati verranno memorizzati nella cartella specificata.
`--resolve`	Scarica anche le dipendenze del pacchetto.

pacchetto `psmisc`

Il pacchetto `psmisc` contiene utilità per la gestione dei processi di sistema:

- `pstree`: il comando `pstree` visualizza i processi correnti sul sistema in una struttura ad albero.
- `killall`: il comando `killall` invia un segnale di kill a tutti i processi identificati dal nome.
- `fuser`: il comando `fuser` Identifica il `PID` di processi che utilizzano i file o i file system specificati.

Esempi:

```
Questo comando è molto utile per creare rapidamente un repository locale di
alcuni rpm!
```

```
# killall httpd
```

Arresta i processi (opzione `-k`) che accedono al file `/etc/httpd/conf/httpd.conf`:

```
# fuser -k /etc/httpd/conf/httpd.conf
```

comando `watch`

Il comando `watch` esegue regolarmente un comando e visualizza il risultato nel terminale a schermo intero.

L'opzione `-n` consente di specificare il numero di secondi tra ogni esecuzione del comando.

> **Nota**
>
> Per uscire dal comando `watch', è necessario digitare i tasti: `CTRL` + `C` per terminare il processo.

Esempi:

- Visualizza la fine del file `/etc/passwd` ogni 5 secondi:

```
$ watch -n 5 tail -n 3 /etc/passwd
```

Risultato:

```
Every 5.0s: tail -n 3 /etc/passwd
rockstar.rockylinux.lan: Thu Jul  1 15:43:59 2021

sssd:x:996:993:User for sssd:/:/sbin/nologin
chrony:x:995:992::/var/lib/chrony:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
```

- Monitoraggio del numero di file in una cartella:

```
$ watch -n 1 'ls -l | wc -l'
```

- Mostra un orologio:

```
$ watch -t -n 1 date
```

Ultimo aggiornamento: 23 dicembre 2022