Command line kung fu

Bash, filtri & co.

Emanuele Santoro
manu@santoro.tk

Corsi GNU/Linux Avanzati 2015

“Il terminale? Nel terzo millennio?”

Anche nell’era delle gui graficose e colorate, usare un interprete di comandi testuale ha i suoi vantaggi

- Funziona su qualsiasi distribuzione di GNU/Linux, e anche sulla maggior parte dei sistemi non-linux
- Si può usare da remoto e/o su una macchina senza schermo
- Molto semplice da programmare, ottimo per automatizzare compiti ripetitivi
  - Essendo un vero e proprio linguaggio di programmazione, non è ambiguo. Se dovete automatizzare operazioni su molti files, è un interprete preciso e semplice da programmare.
- Se nulla altro sembra andare, un terminale non si nega a nessuno

“Il terminale? Nel terzo millennio?”

Anche nell’era delle gui graficose e colorate, usare un interprete di comandi testuale ha i suoi vantaggi

- Funziona su qualsiasi distribuzione di GNU/Linux, e anche sulla maggior parte dei sistemi non-linux
- Si può usare da remoto e/o su una macchina senza schermo
- Molto semplice da programmare, ottimo per automatizzare compiti ripetitivi
    - Essendo un vero e proprio linguaggio di programmazione, non è ambiguo. Se dovete automatizzare operazioni su molti files, è un interprete preciso e semplice da programmare.
- Se nulla altro sembra andare, un terminale non si nega a nessuno

“Il terminale? Nel terzo millennio?”

Anche nell’era delle gui graficose e colorate, usare un interprete di comandi testuale ha i suoi vantaggi

- Funziona su qualsiasi distribuzione di GNU/Linux, e anche sulla maggior parte dei sistemi non-linux
- Si può usare da remoto e/o su una macchina senza schermo
- Molto semplice da programmare, ottimo per automatizzare compiti ripetitivi
    - Essendo un vero e proprio linguaggio di programmazione, non è ambiguo. Se dovete automatizzare operazioni su molti files, è un interprete preciso e semplice da programmare.
- Se nulla altro sembra andare, un terminale non si nega a nessuno

“Il terminale? Nel terzo millennio?”

Anche nell’era delle gui graficose e colorate, usare un interprete di comandi testuale ha i suoi vantaggi

- Funziona su qualsiasi distribuzione di GNU/Linux, e anche sulla maggior parte dei sistemi non-linux
- Si può usare da remoto e/o su una macchina senza schermo
- Molto semplice da programmare, ottimo per automatizzare compiti ripetitivi
  - Essendo un vero e proprio linguaggio di programmazione, non è ambiguo. Se dovete automatizzare operazioni su molti files, è un interprete preciso e semplice da programmare.
- Se nulla altro sembra andare, un terminale non si nega a nessuno

“Il terminale? Nel terzo millennio?”

Anche nell’era delle gui graficose e colorate, usare un interprete di comandi testuale ha i suoi vantaggi

- Funziona su qualsiasi distribuzione di GNU/Linux, e anche sulla maggior parte dei sistemi non-linux
- Si può usare da remoto e/o su una macchina senza schermo
- Molto semplice da programmare, ottimo per automatizzare compiti ripetitivi
  - Essendo un vero e proprio linguaggio di programmazione, non è ambiguo. Se dovete automatizzare operazioni su molti files, è un interprete preciso e semplice da programmare.
- Se nulla altro sembra andare, un terminale non si nega a nessuno

I have no idea what I'm doing

`man`

- Se incontrate un comando nuovo e/o volete saperne di più sulle funzioni di un comando usate `man` (manual)
- La sintassi del comando sarà del tipo `comando [opzioni] [argomenti] argomento`. Tutto ciò che è tra parentesi quadre è opzionale, le opzioni si passano con uno o due trattini (- o --)
- Per cercare all'interno della man page basta scrivere `/terminecercato` e premere `Enter`. Premendo n si passa al prossimo match.
  - La ricerca non fa il wrap-around, arrivati alla fine della manpage premete `p` per tornare in cima
- Premendo `q` si esce dalla manpage
- `Se avete dubbi... man man ;)`

I have no idea what I'm doing

`man`

- Se incontrate un comando nuovo e/o volete saperne di più sulle funzioni di un comando usate `man` (manual)
- La sintassi del comando sarà del tipo `comando [opzioni] [argomenti] argomento`. Tutto ciò che è tra parentesi quadre è opzionale, le opzioni si passano con uno o due trattini (- o --)
- Per cercare all'interno della man page basta scrivere `/terminecercato` e premere `Enter`. Premendo `n` si passa al prossimo match.
  - La ricerca non fa il wrap-around, arrivati alla fine della manpage premete `p` per tornare in cima
- Premendo `q` si esce dalla manpage
- `Se avete dubbi... man man ;)`

I have no idea what I'm doing

`man`

- Se incontrate un comando nuovo e/o volete saperne di più sulle funzioni di un comando usate `man` (manual)
- La sintassi del comando sarà del tipo `comando [opzioni] [argomenti] argomento`. Tutto ciò che è tra parentesi quadre è opzionale, le opzioni si passano con uno o due trattini (- o --)
- Per cercare all'interno della man page basta scrivere `/terminecercato` e premere `Enter`. Premendo n si passa al prossimo match.
  - La ricerca non fa il wrap-around, arrivati alla fine della manpage premete `p` per tornare in cima
- Premendo `q` si esce dalla manpage
- `Se avete dubbi... man man ;)`

I have no idea what I'm doing

`man`

- Se incontrate un comando nuovo e/o volete saperne di più sulle funzioni di un comando usate `man` (manual)
- La sintassi del comando sarà del tipo `comando [opzioni] [argomenti] argomento`. Tutto ciò che è tra parentesi quadre è opzionale, le opzioni si passano con uno o due trattini (- o --)
- Per cercare all'interno della man page basta scrivere `/terminecercato` e premere `Enter`. Premendo n si passa al prossimo match.
  - La ricerca non fa il wrap-around, arrivati alla fine della manpage premete `p` per tornare in cima
- Premendo q si esce dalla manpage
- Se avete dubbi... `man man` ;)

I have no idea what I'm doing

`man`

- Se incontrate un comando nuovo e/o volete saperne di più sulle funzioni di un comando usate `man` (manual)
- La sintassi del comando sarà del tipo `comando [opzioni] [argomenti] argomento`. Tutto ciò che è tra parentesi quadre è opzionale, le opzioni si passano con uno o due trattini (- o --)
- Per cercare all'interno della man page basta scrivere `/terminecercato` e premere `Enter`. Premendo n si passa al prossimo match.
  - La ricerca non fa il wrap-around, arrivati alla fine della manpage premete `p` per tornare in cima
- Premendo `q` si esce dalla manpage
- `Se avete dubbi... man man ;)`

I have no idea what I'm doing

`man`

- Se incontrate un comando nuovo e/o volete saperne di più sulle funzioni di un comando usate `man` (manual)
- La sintassi del comando sarà del tipo `comando [opzioni] [argomenti] argomento`. Tutto ciò che è tra parentesi quadre è opzionale, le opzioni si passano con uno o due trattini (- o --)
- Per cercare all'interno della man page basta scrivere `/terminecercato` e premere `Enter`. Premendo n si passa al prossimo match.
  - La ricerca non fa il wrap-around, arrivati alla fine della manpage premete `p` per tornare in cima
- Premendo `q` si esce dalla manpage
- `Se avete dubbi... man man ;)`

Muovere i primi passi

- `ls` – elenca i file nella cartella corrente
  - `ls nomecartella` -- elenca i file della cartella con quel nome
  - `-lah` – mostra i file incolonnati con maggiori informazioni (-l) includendo anche i file nascosti (-a) con le dimensioni in formato human-readable (-h)
- `cd nomecartella` – ci sposta nella directory
  - . è la directory corrente, .. è la directory un livello più in su
  - ~ è la cartella home dell'utente corrente ~<utente> è la home di <utente>
- `pwd` – stampa la directory in cui ci si trova

Muovere i primi passi

- `ls` – elenca i file nella cartella corrente
  - `ls nomecartella` -- elenca i file della cartella con quel nome
  - `-lah` – mostra i file incolonnati con maggiori informazioni (-l) includendo anche i file nascosti (-a) con le dimensioni in formato human-readable (-h)
- `cd nomecartella` – ci sposta nella directory
  - . è la directory corrente, .. è la directory un livello più in su
  - ~ è la cartella home dell'utente corrente ~<utente> è la home di <utente>
- `pwd` – stampa la directory in cui ci si trova

Muovere i primi passi

- `ls` – elenca i file nella cartella corrente
  - `ls nomecartella` -- elenca i file della cartella con quel nome
  - `-lah` – mostra i file incolonnati con maggiori informazioni (-l) includendo anche i file nascosti (-a) con le dimensioni in formato human-readable (-h)
- `cd nomecartella` – ci sposta nella directory
  - . è la directory corrente, .. è la directory un livello più in su
  - ~ è la cartella home dell'utente corrente ~<utente> è la home di <utente>
- `pwd` – stampa la directory in cui ci si trova

Creiamo qualcosa

- `touch nomefile` – crea un file vuoto se non esiste
- `mkdir nomecartella` – crea una directory vuota
  - -p – crea tutte le directory necessarie (e.g. `mkdir a/b/c/` crea anche a e b se non esistono)

Nota importante

Il filesystem Linux, a differenza di quello Windows, è case sensitive.
Se create (o modificate) i file ricordate che `foo` $\neq$ `Foo` $\neq$ `FOO`

Creiamo qualcosa

- `touch nomefile` – crea un file vuoto se non esiste
- `mkdir nomecartella` – crea una directory vuota
  - -p – crea tutte le directory necessarie (e.g. `mkdir a/b/c/` crea anche a e b se non esistono)

Nota importante

Il filesystem Linux, a differenza di quello Windows, è case sensitive.
Se create (o modificate) i file ricordate che `foo` $\neq$ `Foo` $\neq$ `FOO`

Creiamo qualcosa

- `touch nomefile` – crea un file vuoto se non esiste
- `mkdir nomecartella` – crea una directory vuota
  - -p – crea tutte le directory necessarie (e.g. `mkdir a/b/c/` crea anche a e b se non esistono)

Nota importante

Il filesystem Linux, a differenza di quello Windows, è case sensitive.
Se create (o modificate) i file ricordate che `foo` $\neq$ `Foo` $\neq$ `FOO`

Io ti ho creato, io ti distruggo

- `rm nomefile` – cancella un file
  - `-r` – se il file è una cartella, cancella ricorsivamente il suo contenuto
- `rmdir nomecartella` – cancella una cartella solo se vuota

Attenzione

Nel terminale, non esiste un "cestino". Una volta che avete rimosso un file, è perso per sempre.

Io ti ho creato, io ti distruggo

- `rm nomefile` – cancella un file
  - `-r` – se il file è una cartella, cancella ricorsivamente il suo contenuto
- `rmdir nomecartella` – cancella una cartella solo se vuota

Attenzione

Nel terminale, non esiste un "cestino". Una volta che avete rimosso un file, è perso per sempre.

Io ti ho creato, io ti distruggo

- `rm nomefile` – cancella un file
  - `-r` – se il file è una cartella, cancella ricorsivamente il suo contenuto
- `rmdir nomecartella` – cancella una cartella solo se vuota

Attenzione

Nel terminale, non esiste un "cestino". Una volta che avete rimosso un file, è perso per sempre.

Modifiche di base ai file

- `cp sorgente destinazione` – Copia il file `sorgente` nel file `destinazione`
  - `-r` – Da usare per copiare una directory
- `mv sorgente destinazione` – Sposta il file sorgente in destinazione
  - Si può usare per rinominare i file se `destinazione` è nella stessa cartella di `sorgente`
- `ln -s sorgente /path/al/collegamento` – Crea un collegamento simbolico al file `sorgente` in `/path/al/collegamento`

Modifiche di base ai file

- `cp sorgente destinazione` – Copia il file `sorgente` nel file `destinazione`
  - `-r` – Da usare per copiare una directory
- `mv sorgente destinazione` – Sposta il file sorgente in destinazione
  - Si può usare per rinominare i file se `destinazione` è nella stessa cartella di `sorgente`
- `ln -s sorgente /path/al/collegamento` – Crea un collegamento simbolico al file `sorgente` in `/path/al/collegamento`

Modifiche di base ai file

- `cp sorgente destinazione` – Copia il file `sorgente` nel file `destinazione`
  - `-r` – Da usare per copiare una directory
- `mv sorgente destinazione` – Sposta il file sorgente in destinazione
  - Si può usare per rinominare i file se `destinazione` è nella stessa cartella di `sorgente`
- `ln -s sorgente /path/al/collegamento` – Crea un collegamento simbolico al file `sorgente` in `/path/al/collegamento`

Historia magistra vitae

- `Ctrl+p` o ↑ per andare indietro (comandi più vecchi) nella history dei comandi
- `Ctrl+n` o ↓ per avanti (comandi più recenti) nella history dei comandi
- `Ctrl+r` per effetturare la ricerca di una stringa nei comandi della history
- La history è salvata nel file `.bash_history` nella home dell'utente
- Il comando `history` vi mostra ... la vostra history dei comandi

Historia magistra vitae

- `Ctrl+p` o ↑ per andare indietro (comandi più vecchi) nella history dei comandi
- `Ctrl+n` o ↓ per avanti (comandi più recenti) nella history dei comandi
- `Ctrl+r` per effetturare la ricerca di una stringa nei comandi della history
- La history è salvata nel file `.bash_history` nella home dell'utente
- Il comando `history` vi mostra ... la vostra history dei comandi

Historia magistra vitae

- `Ctrl+p` o ↑ per andare indietro (comandi più vecchi) nella history dei comandi
- `Ctrl+n` o ↓ per avanti (comandi più recenti) nella history dei comandi
- `Ctrl+r` per effetturare la ricerca di una stringa nei comandi della history
- La history è salvata nel file `.bash_history` nella home dell'utente
- Il comando `history` vi mostra ... la vostra history dei comandi

Historia magistra vitae

- `Ctrl+p` o ↑ per andare indietro (comandi più vecchi) nella history dei comandi
- `Ctrl+n` o ↓ per avanti (comandi più recenti) nella history dei comandi
- `Ctrl+r` per effetturare la ricerca di una stringa nei comandi della history
- La history è salvata nel file `.bash_history` nella home dell'utente
- Il comando `history` vi mostra ... la vostra history dei comandi

Historia magistra vitae

- `Ctrl+p` o ↑ per andare indietro (comandi più vecchi) nella history dei comandi
- `Ctrl+n` o ↓ per avanti (comandi più recenti) nella history dei comandi
- `Ctrl+r` per effetturare la ricerca di una stringa nei comandi della history
- La history è salvata nel file `.bash_history` nella home dell'utente
- Il comando `history` vi mostra ... la vostra history dei comandi

Domande?

Demo!

Domande?

Demo!

Text-editor e le religioni

Nell'amministrazione di sistemi Unix e Unix-compatibili si tende a passare molto tempo usando editor di testo.
Viene pertanto naturale abituarsi ad un certo editor, personalizzarlo, renderlo più efficace.
Gli editor principali su i sistemi Unix sono tre:

- `nano`
- `vi`
- `GNU Emacs`

Nano è un editor semplice da usare, che offre poche possibilità di personalizzazione e decisamente poche funzionalità avanzate.
vi e GNU Emacs sono due editor complessi ~~*e il loro mistero è superato solo dalla loro potenza*~~ ma molto efficaci.

vi for dummies

Perché imparare ad usare vi, anche se ci piace GNU Emacs, nano, o qualcos'altro?
Senza entrare nelle guerre di religione, vi è l'unico editor che sicuramente troverete in tutti gli Unix che vi troverete ad usare. Infatti la sua presenza è "imposta" nella "Single UNIX specification".

editor modale

Vi è un editor “particolare”.
Vi è un editor “modale”, nel senso che ha due modalità di funzionamento, e gli stessi tasti si comportano diversamente a seconda della modalità in cui vi trovate.
Le due modalità sono la “modalità inserimento” e la “modalità comandi”.
La modalità predefinita è la modalità comandi. Premendo Esc si torna sempre nella modalità comandi.
Premendo i, a oppure o si entra nella modalità inserimento.

modalità comandi

Nella modalità comandi, tutti i tasti della vostra tastiera forniscono dei comandi all'editor. Sequenze diverse di caratteri forniscono comandi diversi.

- Spostarsi nel file: h (sinistra) j (sotto) k (sopra) l (destra)
- i – inserire del testo a partire dalla posizione corrente del cursore
- / – inizia una ricerca di testo nel documento
- n – trova la prossima occorrenza del testo cercato nella ricerca
- u – annulla l'ultima operazione (undo)
- . – ripete l'ultima operazione (eventualmente annulla l'annullamento)
- Esc : (Esc, poi due punti) – comincia ad inserire un comando

Vi for dummies - i comandi

- :w – write, salva il file corrente
- :q – quit (esci)
- I comandi possono essere concatenati: salva + esci = :wq

A seconda del tipo di vi che state usando (vim, nvi, elvis o altro) ci potrebbero essere dalle decine alle centinaia di altri comandi, ma questi sono bene o male quelli che vi serviranno sempre.

Vi for dummies - la modalità inserimento

Nella modalità inserimento si può effettivamente inserire/cancellare del testo.
Per entrare in modalità inserimento si preme:

- i – per inserire del testo nella posizione corrente del cursore (insert)

Una volta fatte le modifiche desiderate, si può tornare alla modalità comandi premendo Esc.

Demo!

Demo!

Bash++

- `cat` – concatena e stampa file su schermo
- `less` – stampa file su schermo con una finestra scorrevole, come la manpage
- `echo` – stampa il valore di un'espressione
- `locate` – cerca file in un database
  - Il database va aggiornato periodicamente con il comando `updatedb`
- `find` – cerca dei file all'interno di una gerarchia di cartelle, eventualmente con dei parametri di ricerca

Bash++

- `cat` – concatena e stampa file su schermo
- `less` – stampa file su schermo con una finestra scorrevole, come la manpage
- `echo` – stampa il valore di un'espressione
- `locate` – cerca file in un database
  - Il database va aggiornato periodicamente con il comando `updatedb`
- `find` – cerca dei file all'interno di una gerarchia di cartelle, eventualmente con dei parametri di ricerca

Bash++

- `cat` – concatena e stampa file su schermo
- `less` – stampa file su schermo con una finestra scorrevole, come la manpage
- `echo` – stampa il valore di un'espressione
- `locate` – cerca file in un database
  - Il database va aggiornato periodicamente con il comando `updatedb`
- `find` – cerca dei file all'interno di una gerarchia di cartelle, eventualmente con dei parametri di ricerca

Bash++

- `cat` – concatena e stampa file su schermo
- `less` – stampa file su schermo con una finestra scorrevole, come la manpage
- `echo` – stampa il valore di un'espressione
- `locate` – cerca file in un database
  - Il database va aggiornato periodicamente con il comando `updatedb`
- `find` – cerca dei file all'interno di una gerarchia di cartelle, eventualmente con dei parametri di ricerca

Bash++

- `cat` – concatena e stampa file su schermo
- `less` – stampa file su schermo con una finestra scorrevole, come la manpage
- `echo` – stampa il valore di un'espressione
- `locate` – cerca file in un database
  - Il database va aggiornato periodicamente con il comando `updatedb`
- `find` – cerca dei file all'interno di una gerarchia di cartelle, eventualmente con dei parametri di ricerca

Bash++

- `cat` – concatena e stampa file su schermo
- `less` – stampa file su schermo con una finestra scorrevole, come la manpage
- `echo` – stampa il valore di un'espressione
- `locate` – cerca file in un database
  - Il database va aggiornato periodicamente con il comando `updatedb`
- `find` – cerca dei file all'interno di una gerarchia di cartelle, eventualmente con dei parametri di ricerca

Bash++

- `cat` – concatena e stampa file su schermo
- `less` – stampa file su schermo con una finestra scorrevole, come la manpage
- `echo` – stampa il valore di un'espressione
- `locate` – cerca file in un database
  - Il database va aggiornato periodicamente con il comando `updatedb`
- `find` – cerca dei file all'interno di una gerarchia di cartelle, eventualmente con dei parametri di ricerca

Standard streams

Ogni processo ha almeno 3 canali di comunicazione di default:

- stdin (0) – il canale che di riceve l'input, di default riceve ciò che l'utente scrive sul terminale
- stdout (1) – il canale che stampa l'output del programma, di default stampa su terminale
- stderr (2) – un canale extra per segnalare errori senza mischiarli con l'output. di default scrive sul terminale

Con pipe e redirezioni si possono connettere in vari modi questi canali, creando delle vere e proprie "catene di montaggio" di comandi bash che permettono di risolvere molti problemi unendo tanti blocchi semplici.

Standard streams

Ogni processo ha almeno 3 canali di comunicazione di default:

- stdin (0) – il canale che di riceve l'input, di default riceve ciò che l'utente scrive sul terminale
- stdout (1) – il canale che stampa l'output del programma, di default stampa su terminale
- stderr (2) – un canale extra per segnalare errori senza mischiarli con l'output. di default scrive sul terminale

Con pipe e redirezioni si possono connettere in vari modi questi canali, creando delle vere e proprie "catene di montaggio" di comandi bash che permettono di risolvere molti problemi unendo tanti blocchi semplici.

Standard streams

Ogni processo ha almeno 3 canali di comunicazione di default:

- stdin (0) – il canale che di riceve l'input, di default riceve ciò che l'utente scrive sul terminale
- stdout (1) – il canale che stampa l'output del programma, di default stampa su terminale
- stderr (2) – un canale extra per segnalare errori senza mischiarli con l'output. di default scrive sul terminale

Con pipe e redirezioni si possono connettere in vari modi questi canali, creando delle vere e proprie "catene di montaggio" di comandi bash che permettono di risolvere molti problemi unendo tanti blocchi semplici.

Standard streams

Ogni processo ha almeno 3 canali di comunicazione di default:

- stdin (0) – il canale che di riceve l'input, di default riceve ciò che l'utente scrive sul terminale
- stdout (1) – il canale che stampa l'output del programma, di default stampa su terminale
- stderr (2) – un canale extra per segnalare errori senza mischiarli con l'output. di default scrive sul terminale

Con pipe e redirezioni si possono connettere in vari modi questi canali, creando delle vere e proprie "catene di montaggio" di comandi bash che permettono di risolvere molti problemi unendo tanti blocchi semplici.

Standard streams

Ogni processo ha almeno 3 canali di comunicazione di default:

- stdin (0) – il canale che di riceve l'input, di default riceve ciò che l'utente scrive sul terminale
- stdout (1) – il canale che stampa l'output del programma, di default stampa su terminale
- stderr (2) – un canale extra per segnalare errori senza mischiarli con l'output. di default scrive sul terminale

Con pipe e redirezioni si possono connettere in vari modi questi canali, creando delle vere e proprie "catene di montaggio" di comandi bash che permettono di risolvere molti problemi unendo tanti blocchi semplici.

Redirezione

- `comando < file` – connette stdin di un processo ad un file
- `comando > file` – connette stdout di un processo ad un file. Se il file esiste viene cancellato e sovrascritto
- `comando 2> file` – connette stderr di un processo ad un file. Se il file esiste viene cancellato e sovrascritto
- comando `&> file` – connette stdout e stderr di un processo ad un file. Se il file esiste viene cancellato e sovrascritto
- Utilizzando >> al posto di > in uno degli ultimi 3 comandi si ottiene lo stesso risultato ma il risultato viene aggiunto al file se esistente
- `comando1 | comando2` – lo stdout di comando1 diventa lo stdin di comando2

Redirezione

- `comando < file` – connette stdin di un processo ad un file
- `comando > file` – connette stdout di un processo ad un file. Se il file esiste viene cancellato e sovrascritto
- `comando 2> file` – connette stderr di un processo ad un file. Se il file esiste viene cancellato e sovrascritto
- comando `&> file` – connette stdout e stderr di un processo ad un file. Se il file esiste viene cancellato e sovrascritto
- Utilizzando >> al posto di > in uno degli ultimi 3 comandi si ottiene lo stesso risultato ma il risultato viene aggiunto al file se esistente
- `comando1 | comando2` – lo stdout di comando1 diventa lo stdin di comando2

Redirezione

- `comando < file` – connette stdin di un processo ad un file
- `comando > file` – connette stdout di un processo ad un file. Se il file esiste viene cancellato e sovrascritto
- `comando 2> file` – connette stderr di un processo ad un file. Se il file esiste viene cancellato e sovrascritto
- comando `&> file` – connette stdout e stderr di un processo ad un file. Se il file esiste viene cancellato e sovrascritto
- Utilizzando >> al posto di > in uno degli ultimi 3 comandi si ottiene lo stesso risultato ma il risultato viene aggiunto al file se esistente
- `comando1 | comando2` – lo stdout di comando1 diventa lo stdin di comando2

Redirezione

- `comando < file` – connette stdin di un processo ad un file
- `comando > file` – connette stdout di un processo ad un file. Se il file esiste viene cancellato e sovrascritto
- `comando 2> file` – connette stderr di un processo ad un file. Se il file esiste viene cancellato e sovrascritto
- comando `&> file` – connette stdout e stderr di un processo ad un file. Se il file esiste viene cancellato e sovrascritto
- Utilizzando >> al posto di > in uno degli ultimi 3 comandi si ottiene lo stesso risultato ma il risultato viene aggiunto al file se esistente
- `comando1 | comando2` – lo stdout di comando1 diventa lo stdin di comando2

Redirezione

- `comando < file` – connette stdin di un processo ad un file
- `comando > file` – connette stdout di un processo ad un file. Se il file esiste viene cancellato e sovrascritto
- `comando 2> file` – connette stderr di un processo ad un file. Se il file esiste viene cancellato e sovrascritto
- comando `&> file` – connette stdout e stderr di un processo ad un file. Se il file esiste viene cancellato e sovrascritto
- Utilizzando $>>$ al posto di $>$ in uno degli ultimi 3 comandi si ottiene lo stesso risultato ma il risultato viene aggiunto al file se esistente
- `comando1 | comando2` – lo stdout di comando1 diventa lo stdin di comando2

Redirezione

- `comando < file` – connette stdin di un processo ad un file
- `comando > file` – connette stdout di un processo ad un file. Se il file esiste viene cancellato e sovrascritto
- `comando 2> file` – connette stderr di un processo ad un file. Se il file esiste viene cancellato e sovrascritto
- comando `&> file` – connette stdout e stderr di un processo ad un file. Se il file esiste viene cancellato e sovrascritto
- Utilizzando $>>$ al posto di $>$ in uno degli ultimi 3 comandi si ottiene lo stesso risultato ma il risultato viene aggiunto al file se esistente
- `comando1 | comando2` – lo stdout di comando1 diventa lo stdin di comando2

Esecuzione condizionale

- `comando1 && comando2` – esegue `comando2` se e solo se `comando1` ha successo (codice di ritorno $= 0$)
- `comando1 || comando2` – esegue `comando2` se e solo se `comando1` fallisce (codice di ritorno $\neq 0$)

Esecuzione condizionale

- `comando1 && comando2` – esegue `comando2` se e solo se `comando1` ha successo (codice di ritorno $= 0$)
- `comando1 || comando2` – esegue `comando2` se e solo se `comando1` fallisce (codice di ritorno $\neq 0$)

Cos'è un filtro ?

In Unix, un filtro è un programma che:

- legge dati in ingresso sullo stdin
- trasforma questi dati in qualche maniera
- produce dell'output su stdout. l'ouput può essere una versione trasformata dell'input oppure altro.

In Unix i filtri sono importanti perché data la loro natura (leggere su stdin e scrivere su stdout) si prestano molto bene ad essere usati "in cascata" tramite le pipe.
Esempio: un programma scarica una pagina web e lo passa ad un altro programma che estrae degli URL e li passa ad un altro programma che scarica tali url. (hint: curl, grep, xargs+youtube-dl)

Alcuni filtri

- `cut`
- `sort`
- `uniq`
- `wc`
- `tee`
- `head e tail`
- `grep`

cut

- Estrae colonne delimitate da un carattere speciale da ogni riga di un file
- `-d` – specifica il delimitatore (default Tab)
- `-f` – specifica quale colonna estrarre (one-based)

sort

- Ordina le righe di un file
- `-k` – specifica quali colonne del file usare come chiave per l'ordinamento
- `-t` – specifica il delimitatore tra le colonne (default whitespace)

uniq

- Stampa le righe uniche di un file già ordinato
- `-c` – conta le occorrenze
- `-d` – mostra solo i duplicati
- `-u` – mostra solo i non duplicati

`WC`

- Conta righe, parole e caratteri
- `-l` – mostra solo il numero di righe
- `-w` – mostra solo il numero di parole
- `-c` – mostra solo il numero di caratteri

tee

- Connette il suo stdin allo stdin di due o più file
- Utile per mostrare l'output di un comando a schermo e darlo come input ad un altro comando

head e tail

- `head` mostra le prime 10 righe di un file, `tail` le ultime 10
- `-nX` – mostra le prime/ultime X righe
- `tail -f` – permette di "tenere d'occhio" un file a cui vengono continuamente aggiunte righe in coda (ad esempio un log)

grep

grep è uno dei filtri più potenti ed importanti di un sistema Unix.
Mostra le righe di un file che corrispondono (o meno) ad un pattern (regular expression [1])
Grep permette di ricevere in input del testo, una linea alla volta, ed effettuare delle ricerche con opportuni parametri all'interno di ogni linea, e stampare il risultato.

[1] https://xkcd.com/208/

grep (2)

- `-v` – attiva il match invertito (mostra le righe che **non** corrispondono)
- `-i` – ricerca case insensitive
- `-c` – conta i match
- `-l` – mostra solo i nomi dei file con match
- `-r` – ricerca ricorsivamente all'interno dei file a partire da una cartella
- `-E` – usa le extended regular expression
- -P – usa PCRE, la sintassi per le espressioni regolari in stile Perl, presenti nella maggior parte dei linguaggi di programmazione
- -o – mostra solo la parte di testo che matcha la regexp, non tutta la linea (utile per estrarre del testo)

sed

Sed è l'abbreviazione di **s**tream **ed**itor.
Trasforma ogni linea che legge dallo stdin in base ad una serie di istruzioni date come argomenti al programma stesso, e stampa il risultato sullo stdout
L'uso più comune è effettuare delle sostituzioni nel testo:

- echo night | sed s/night/day/
  - day

Ma può fare molto di più.
Per maggiori informazioni consultate
http://www.grymoire.com/Unix/Sed.html

awk

Awk è un filtro che consente di estrapolare ed eventualmente aggregare dati provenienti da un file o da un flusso (connesso allo stdin).
Nella versione più semplice, awk può dividere una serie di linee in formato tabulare ed effettuare delle operazioni su ogni elemento di ogni linea.
Praticamente, un esempio: rilevare tutti i programmi che sta eseguendo un utente:

- ps aux | awk '$1 == "manu" {print $11}'

Quando awk legge una linea, la spezza ad ogni occorrenza di un delimitatore (lo spazio è il delimitatore predefinito, altrimenti è possibile specificare un delimitatore personalizzato con l'opzione -F). Alla variabile $0 viene assegnata l'intera linea, a $1 il primo elemento della linea, a $2 il secondo e così via.

Awk (links)

Effective AWK programming, il manuale della versione GNU di awk (quella distribuita con quasi tutte le distribuzioni di GNU/Linux):

- https://www.gnu.org/software/gawk/manual/

Una buona guida ad awk:

- http://www.grymoire.com/Unix/Awk.html

Domande?

Demo!

Domande?

Demo!

Shell scripting

Una serie di comandi di shell possono essere salvati in un file di testo per essere eseguiti nuovamente senza doverli digitare nuovamente. Un file che contiene comandi di shell è uno shell-script.

```
#!/bin/bash
echo "Hello, world!"
```

Anatomia di uno script

Uno script in Unix ha sempre lo shabang come prima linea: indica all'interprete dei comandi con quale programma interpretare il resto del file:

```
#!  <path dell'interprete>
comando1
comando2
comando3
```

Una volta salvato come hello.sh, bisogna renderlo eseguibile:

- `chmod +x hello.sh`

E sarà eseguibile digitando ./hello.sh oppure /path/completo/di/hello.sh

Anatomia di uno script

Uno script in Unix ha sempre lo shabang come prima linea: indica all'interprete dei comandi con quale programma interpretare il resto del file:

```
#!  <path dell'interprete>
comando1
comando2
comando3
```

Una volta salvato come hello.sh, bisogna renderlo eseguibile:

- `chmod +x hello.sh`

E sarà eseguibile digitando ./hello.sh oppure /path/completo/di/hello.sh

Anatomia di uno script

Uno script in Unix ha sempre lo shabang come prima linea: indica all'interprete dei comandi con quale programma interpretare il resto del file:

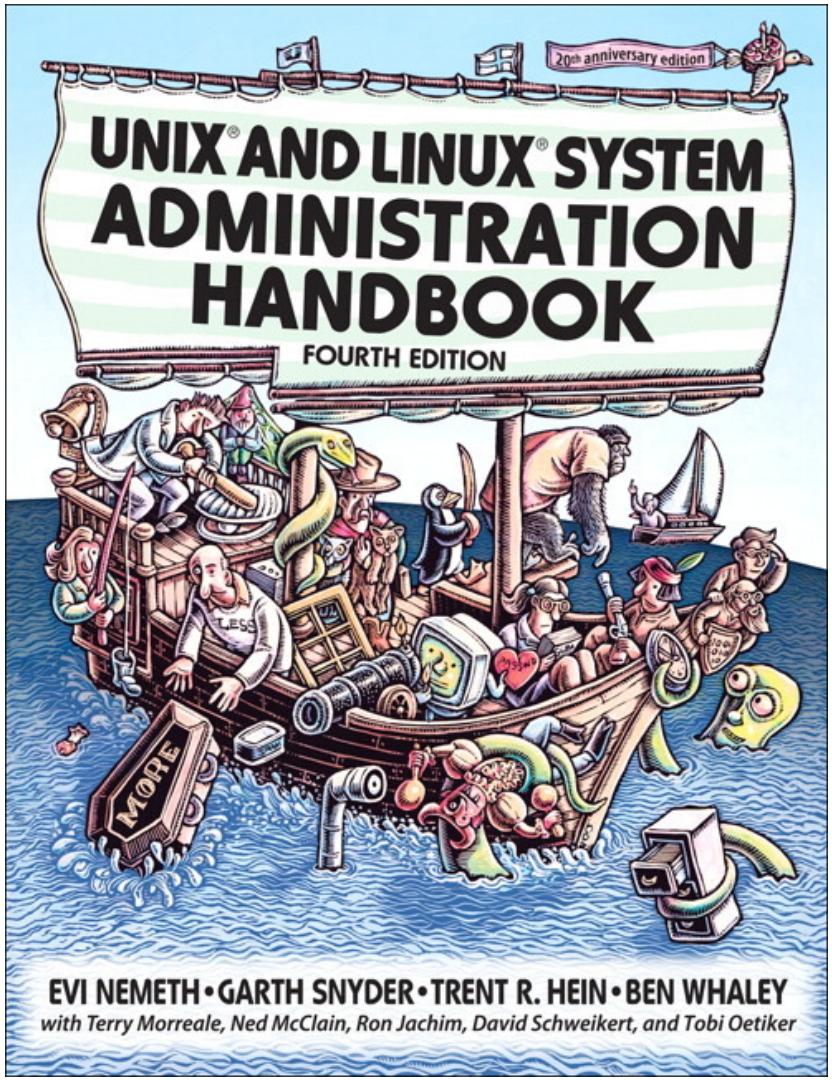
```
#!  <path dell'interprete>
comando1
comando2
comando3
```

Una volta salvato come hello.sh, bisogna renderlo eseguibile:

- `chmod +x hello.sh`

E sarà eseguibile digitando ./hello.sh oppure /path/completo/di/hello.sh

Per approfondire



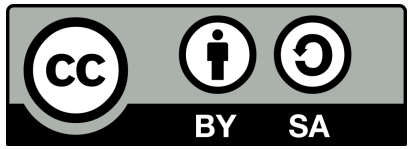
Google is your friend

Last call

Domande?

Goodbye and thanks for all the fish

Grazie per l'attenzione!



`http://www.poul.org`