Command line kung fu

Bash, filtri & co.

Giulio De Pasquale
giulio@depasquale.eu

Corsi Linux Avanzati 2016

“Il terminale? Nel 2016?”

- Universale: presente su ogni distribuzione GNU/Linux.
- Si può usare da remoto e/o su una macchina senza schermo.
- Consente di automatizzare compiti ripetitivi
  - Ad esempio, permette di operare in maniera precisa e veloce su molti files.

"Il terminale? Nel 2016?"

- Universale: presente su ogni distribuzione GNU/Linux.
- Si può usare da remoto e/o su una macchina senza schermo.
- Consente di automatizzare compiti ripetitivi
  - Ad esempio, permette di operare in maniera precisa e veloce su molti files.

“Il terminale? Nel 2016?”

- Universale: presente su ogni distribuzione GNU/Linux.
- Si può usare da remoto e/o su una macchina senza schermo.
- Consente di automatizzare compiti ripetitivi
  - Ad esempio, permette di operare in maniera precisa e veloce su molti files.

“Il terminale? Nel 2016?”

- Universale: presente su ogni distribuzione GNU/Linux.
- Si può usare da remoto e/o su una macchina senza schermo.
- Consente di automatizzare compiti ripetitivi
  - Ad esempio, permette di operare in maniera precisa e veloce su molti files.

“Struttura di un comando”

`comando [opzioni] [argomenti]`

- Tutto ciò che è tra parentesi quadre è facoltativo.
- Le opzioni si passano con uno o due trattini (- o --)

Esempio

$$\underbrace{\textit{cuocipizza}}_{\textit{comando}}\ \underbrace{\textit{--fornoalegna}}_{\textit{opzione}}\ \underbrace{\textit{margherita}}_{\textit{argomento}}\ \underbrace{\textit{capricciosa}}_{\textit{argomento}}$$

"Struttura di un comando"

`comando [opzioni] [argomenti]`

- Tutto ciò che è tra parentesi quadre è facoltativo.
- Le opzioni si passano con uno o due trattini (- o --)

Esempio

$\underbrace{\textit{cuocipizza}}_{\textit{comando}}\ \underbrace{\textit{--fornoalegna}}_{\textit{opzione}}\ \underbrace{\textit{margherita}}_{\textit{argomento}}\ \underbrace{\textit{capricciosa}}_{\textit{argomento}}$

"Struttura di un comando"

`comando [opzioni] [argomenti]`

- Tutto ciò che è tra parentesi quadre è facoltativo.
- Le opzioni si passano con uno o due trattini (- o --)

Esempio

$$\underbrace{\textit{cuocipizza}}_{\textit{comando}}\ \underbrace{-\textit{fornoalegna}}_{\textit{opzione}}\ \underbrace{\textit{margherita}}_{\textit{argomento}}\ \underbrace{\textit{capricciosa}}_{\textit{argomento}}$$

Shortcut di base

- `TAB`
  - Autocompletamento comandi
- `CTRL+C`
  - Cerca di interrompere il processo in esecuzione
- `SHIFT+CTRL+C / V`
  - Copia / Incolla

Shortcut di base

- `TAB`
  - Autocompletamento comandi
- `CTRL+C`
  - Cerca di interrompere il processo in esecuzione
- `SHIFT+CTRL+C / V`
  - Copia / Incolla

Shortcut di base

- `TAB`
  - Autocompletamento comandi
- `CTRL+C`
  - Cerca di interrompere il processo in esecuzione
- `SHIFT+CTRL+C / V`
  - Copia / Incolla

Shortcut di base

- `TAB`
  - Autocompletamento comandi
- `CTRL+C`
  - Cerca di interrompere il processo in esecuzione
- `SHIFT+CTRL+C / V`
  - Copia / Incolla

Shortcut di base

- `TAB`
  - Autocompletamento comandi
- `CTRL+C`
  - Cerca di interrompere il processo in esecuzione
- `SHIFT+CTRL+C / V`
  - Copia / Incolla

Shortcut di base

- `TAB`
  - Autocompletamento comandi
- `CTRL+C`
  - Cerca di interrompere il processo in esecuzione
- `SHIFT+CTRL+C / V`
  - Copia / Incolla

I have no idea what i'm doing

`man`

Permette di leggere la documentazione (se presente) di un comando passato come argomento.

- Per cercare all'interno della man page basta scrivere `/terminecercato` e premere *Enter*. Premendo *n* si passa al prossimo risultato.
- Premendo *p* si torna in cima.
- Premendo q si esce dal manuale.

In caso di emergenza

$\underbrace{man}_{comando}$ $\underbrace{man}_{argomento}$

I have no idea what i'm doing

`man`

Permette di leggere la documentazione (se presente) di un comando passato come argomento.

- Per cercare all'interno della man page basta scrivere `/terminecercato` e premere *Enter*. Premendo *n* si passa al prossimo risultato.
- Premendo *p* si torna in cima.
- Premendo q si esce dal manuale.

In caso di emergenza

$\underbrace{man}_{comando}$ $\underbrace{man}_{argomento}$

I have no idea what i'm doing

`man`

Permette di leggere la documentazione (se presente) di un comando passato come argomento.

- Per cercare all'interno della man page basta scrivere `/terminecercato` e premere *Enter*. Premendo *n* si passa al prossimo risultato.
- Premendo *p* si torna in cima.
- Premendo `q` si esce dal manuale.

In caso di emergenza

man *man*

comando *argomento*

I have no idea what i'm doing

`man`

Permette di leggere la documentazione (se presente) di un comando passato come argomento.

- Per cercare all'interno della man page basta scrivere `/terminecercato` e premere *Enter*. Premendo *n* si passa al prossimo risultato.
- Premendo *p* si torna in cima.
- Premendo `q` si esce dal manuale.

In caso di emergenza

$\underbrace{man}_{comando}$ $\underbrace{man}_{argomento}$

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 corrente

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 corrente

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 corrente

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 corrente

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 corrente

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 corrente

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 corrente

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 corrente

Creiamo qualcosa

- `touch nomefile` – crea un file vuoto se non esiste, altrimenti ne aggiorna la data e l'ora dell'ultimo accesso
- `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, altrimenti ne aggiorna la data e l'ora dell'ultimo accesso
- `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, altrimenti ne aggiorna la data e l'ora dell'ultimo accesso
- `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, altrimenti ne aggiorna la data e l'ora dell'ultimo accesso
- `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
- Se siete paranoici: `shred nomefile` – sovrascrive un file con bit random 3 volte
  - `-u` – dopo averlo sovrascritto, lo cancella
  - `-n <k>` – sovrascrive il file k volte invece di tre
  - `-z` – sovrascrive il file un'ultima volta con degli zeri, per nascondere la rimozione sicura

Attenzione

Nel terminale, non esiste un "cestino". Una volta che avete rimosso un file, è perso per sempre.[1]

[1] A meno di strani magheggi con software forense

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
- Se siete paranoici: `shred nomefile` – sovrascrive un file con bit random 3 volte
  - -u – dopo averlo sovrascritto, lo cancella
  - -n <k> – sovrascrive il file k volte invece di tre
  - -z – sovrascrive il file un'ultima volta con degli zeri, per nascondere la rimozione sicura

Attenzione

Nel terminale, non esiste un "cestino". Una volta che avete rimosso un file, è perso per sempre.[1]

[1] A meno di strani magheggi con software forense

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
- Se siete paranoici: `shred nomefile` – sovrascrive un file con bit random 3 volte
  - -u – dopo averlo sovrascritto, lo cancella
  - -n <k> – sovrascrive il file k volte invece di tre
  - -z – sovrascrive il file un'ultima volta con degli zeri, per nascondere la rimozione sicura

Attenzione

Nel terminale, non esiste un "cestino". Una volta che avete rimosso un file, è perso per sempre.[1]

[1] A meno di strani magheggi con software forense

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
- Se siete paranoici: `shred nomefile` – sovrascrive un file con bit random 3 volte
  - -u – dopo averlo sovrascritto, lo cancella
  - -n <k> – sovrascrive il file k volte invece di tre
  - -z – sovrascrive il file un'ultima volta con degli zeri, per nascondere la rimozione sicura

Attenzione

Nel terminale, non esiste un "cestino". Una volta che avete rimosso un file, è perso per sempre.[1]

[1] A meno di strani magheggi con software forense

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
- Se siete paranoici: `shred nomefile` – sovrascrive un file con bit random 3 volte
  - `-u` – dopo averlo sovrascritto, lo cancella
  - -n <k> – sovrascrive il file k volte invece di tre
  - -z – sovrascrive il file un'ultima volta con degli zeri, per nascondere la rimozione sicura

Attenzione

Nel terminale, non esiste un "cestino". Una volta che avete rimosso un file, è perso per sempre.[1]

[1] A meno di strani magheggi con software forense

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
- Se siete paranoici: `shred nomefile` – sovrascrive un file con bit random 3 volte
  - `-u` – dopo averlo sovrascritto, lo cancella
  - -n <k> – sovrascrive il file k volte invece di tre
  - -z – sovrascrive il file un'ultima volta con degli zeri, per nascondere la rimozione sicura

Attenzione

Nel terminale, non esiste un "cestino". Una volta che avete rimosso un file, è perso per sempre.[1]

[1] A meno di strani magheggi con software forense

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
- Se siete paranoici: `shred nomefile` – sovrascrive un file con bit random 3 volte
  - `-u` – dopo averlo sovrascritto, lo cancella
  - -n <k> – sovrascrive il file k volte invece di tre
  - -z – sovrascrive il file un'ultima volta con degli zeri, per nascondere la rimozione sicura

Attenzione

Nel terminale, non esiste un "cestino". Una volta che avete rimosso un file, è perso per sempre.[1]

[1] A meno di strani magheggi con software forense

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
- Se siete paranoici: `shred nomefile` – sovrascrive un file con bit random 3 volte
  - `-u` – dopo averlo sovrascritto, lo cancella
  - -n <k> – sovrascrive il file k volte invece di tre
  - -z – sovrascrive il file un'ultima volta con degli zeri, per nascondere la rimozione sicura

Attenzione

Nel terminale, non esiste un "cestino". Una volta che avete rimosso un file, è perso per sempre.[1]

[1] A meno di strani magheggi con software forense

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`

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`

Gestione dei permessi

In Linux gli utenti appartengono a molteplici gruppi per scopi amministrativi.
In ogni macchina l'utente amministratore è chiamato *root.*
Ogni file appartiene ad un utente e ad un gruppo, i quali possono modificarne i permessi.
L'utente può $\underbrace{\textit{leggere}}_{(r)ead}$ / $\underbrace{\textit{scrivere}}_{(w)rite}$ / $\underbrace{\textit{eseguire}}_{e(x)ecute}$ i propri file e quelli dei gruppi a cui appartiene.

Gestione dei permessi (2)

- `chown utente[:gruppo] FILE` – modifica il proprietario di un file
  - è possibile modificare solo il gruppo, passando come opzione `:gruppo`
  - *-R* – modifica il proprietario ad un'intera cartella
- `chmod permesso FILE` – modifica i permessi di un file
  - il formato dell'opzione `permesso` è scomposto in 3 parti:
    -
    - +/=/-, mutualmente esclusivi
    - r/w/x, componibile

Gestione dei permessi (2)

- `chown utente[:gruppo] FILE` – modifica il proprietario di un file
  - è possibile modificare solo il gruppo, passando come opzione `:gruppo`
  - *-R* – modifica il proprietario ad un'intera cartella
- `chmod permesso FILE` – modifica i permessi di un file
  - il formato dell'opzione `permesso` è scomposto in 3 parti:
    - `u`, `g`, `o`, `a`: compatibile
    - `+`, `-`, `=`: mutualmente esclusivi
    - `r`, `w`, `x`: compatibile

Gestione dei permessi (2)

- `chown utente[:gruppo] FILE` – modifica il proprietario di un file
  - è possibile modificare solo il gruppo, passando come opzione `:gruppo`
  - *-R* – modifica il proprietario ad un'intera cartella
- `chmod permesso FILE` – modifica i permessi di un file
  - il formato dell'opzione `permesso` è scomposto in 3 parti:
    -
    - `+/-/=`, mutualmente esclusivi
    - `r/w/x`, componibile

Gestione dei permessi (2)

- `chown utente[:gruppo] FILE` – modifica il proprietario di un file
  - è possibile modificare solo il gruppo, passando come opzione `:gruppo`
  - *-R* – modifica il proprietario ad un'intera cartella
- `chmod permesso FILE` – modifica i permessi di un file
  - il formato dell'opzione `permesso` è scomposto in 3 parti:
    - $\underbrace{u}_{user}/\underbrace{g}_{group}/\underbrace{o}_{other}$, componibile
    - `+/-/=`, mutualmente esclusivi
    - `r/w/x`, componibile

Esempio

Rendiamo il file "dog.jpg" leggibile e scrivibile per l'utente, leggibile solo per il gruppo e rimuoviamo tutti i permessi per gli altri.

```
chmod ug=r
chmod u+w
chmod o-rwx
```

Gestione dei permessi (2)

- `chown utente[:gruppo] FILE` – modifica il proprietario di un file
  - è possibile modificare solo il gruppo, passando come opzione `:gruppo`
  - *-R* – modifica il proprietario ad un'intera cartella
- `chmod permesso FILE` – modifica i permessi di un file
  - il formato dell'opzione `permesso` è scomposto in 3 parti:
    - $\underbrace{u}_{user}/\underbrace{g}_{group}/\underbrace{o}_{other}$, componibile
    - `+/-/=`, mutualmente esclusivi
    - `r/w/x`, componibile

Esempio

```
Rendiamo il file "dog.jpg" leggibile e scrivibile per l'utente,
leggibile solo per il gruppo e rimuoviamo tutti i permessi per
gli altri.
chmod ug=r
chmod u+w
chmod o-rwx
```

Gestione dei permessi (2)

- `chown utente[:gruppo] FILE` – modifica il proprietario di un file
  - è possibile modificare solo il gruppo, passando come opzione `:gruppo`
  - *-R* – modifica il proprietario ad un'intera cartella
- `chmod permesso FILE` – modifica i permessi di un file
  - il formato dell'opzione `permesso` è scomposto in 3 parti:
    - $\underbrace{u}_{user}/\underbrace{g}_{group}/\underbrace{o}_{other}$, componibile
    - `+/-/=`, mutualmente esclusivi
    - `r/w/x`, componibile

Esempio

```
Rendiamo il file "dog.jpg" leggibile e scrivibile per l'utente,
leggibile solo per il gruppo e rimuoviamo tutti i permessi per
gli altri.
chmod ug=r
chmod u+w
chmod o-rwx
```

Gestione dei permessi (2)

- `chown utente[:gruppo] FILE` – modifica il proprietario di un file
  - è possibile modificare solo il gruppo, passando come opzione `:gruppo`
  - *-R* – modifica il proprietario ad un'intera cartella
- `chmod permesso FILE` – modifica i permessi di un file
  - il formato dell'opzione `permesso` è scomposto in 3 parti:
    - $\underbrace{u}_{user}/\underbrace{g}_{group}/\underbrace{o}_{other}$, componibile
    - +/−/=, mutualmente esclusivi
    - `r/w/x`, componibile

Esempio

```
Rendiamo il file "dog.jpg" leggibile e scrivibile per l'utente,
leggibile solo per il gruppo e rimuoviamo tutti i permessi per
gli altri.
chmod ug=r
chmod u+w
chmod o-rwx
```

Gestione dei permessi (2)

- `chown utente[:gruppo] FILE` – modifica il proprietario di un file
  - è possibile modificare solo il gruppo, passando come opzione `:gruppo`
  - *-R* – modifica il proprietario ad un'intera cartella
- `chmod permesso FILE` – modifica i permessi di un file
  - il formato dell'opzione `permesso` è scomposto in 3 parti:
    - $\underbrace{u}_{user}/\underbrace{g}_{group}/\underbrace{o}_{other}$, componibile
    - `+/-/=`, mutualmente esclusivi
    - `r/w/x`, componibile

Esempio

```
Rendiamo il file "dog.jpg" leggibile e scrivibile per l'utente,
leggibile solo per il gruppo e rimuoviamo tutti i permessi per
gli altri.
chmod ug=r
chmod u+w
chmod o-rwx
```

Gestione dei permessi (2)

- `chown utente[:gruppo] FILE` – modifica il proprietario di un file
  - è possibile modificare solo il gruppo, passando come opzione `:gruppo`
  - *-R* – modifica il proprietario ad un'intera cartella
- `chmod permesso FILE` – modifica i permessi di un file
  - il formato dell'opzione `permesso` è scomposto in 3 parti:
    - $\underbrace{u}_{user}/\underbrace{g}_{group}/\underbrace{o}_{other}$, componibile
    - `+/-/=`, mutualmente esclusivi
    - `r/w/x`, componibile

Esempio

```
Rendiamo il file "dog.jpg" leggibile e scrivibile per l'utente,
leggibile solo per il gruppo e rimuoviamo tutti i permessi per
gli altri.
chmod ug=r
chmod u+w
chmod o-rwx
```

vi for dummies

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 ma molto potenti.

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 ma molto potenti.

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 ma molto potenti.

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 ma molto potenti.

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 ma molto potenti.

Learn the hard way

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
- *a* – inserire del testo appartire dalla posizione immediatamente successiva al 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)
- *:* (due punti) – comincia ad inserire un comando

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
- *a* – inserire del testo appartire dalla posizione immediatamente successiva al 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)
- *:* (due punti) – comincia ad inserire un comando

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
- *a* – inserire del testo appartire dalla posizione immediatamente successiva al 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)
- *:* (due punti) – comincia ad inserire un comando

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
- *a* – inserire del testo appartire dalla posizione immediatamente successiva al 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)
- *:* (due punti) – comincia ad inserire un comando

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
- *a* – inserire del testo appartire dalla posizione immediatamente successiva al 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)
- *:* (due punti) – comincia ad inserire un comando

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
- *a* – inserire del testo appartire dalla posizione immediatamente successiva al 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)
- *:* (due punti) – comincia ad inserire un comando

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
- *a* – inserire del testo appartire dalla posizione immediatamente successiva al 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)
- *:* (due punti) – comincia ad inserire un comando

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
- *a* – inserire del testo appartire dalla posizione immediatamente successiva al 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)
- *:* (due punti) – comincia ad inserire un comando

Comandi

- `:w` – write, salva il file corrente
- `:q` – quit, esci

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.

Concatenazione comandi

I comandi possono essere concatenati!

Salva + Esci = `:wq`

Comandi

- `:w` – write, salva il file corrente
- `:q` – quit, esci

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.

Concatenazione comandi

I comandi possono essere concatenati!

Salva + Esci = `:wq`

Comandi

- `:w` – write, salva il file corrente
- `:q` – quit, esci

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.

Concatenazione comandi

I comandi possono essere concatenati!

Salva + Esci = `:wq`

Comandi

- `:w` – write, salva il file corrente
- `:q` – quit, esci

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.

Concatenazione comandi

I comandi possono essere concatenati!
Salva + Esci = `:wq`

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)
- *a* – per "appendere" del testo, ovvero inserire del testo nella posizione immediatamente successiva a quella del cursore
- *o* – per inserire del testo nella linea successiva a quella corrente

Dopo ognuno di questi comandi si entra nella modalità inserimento. Per uscire dalla modalità inserimento si preme Esc.

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)
- *a* – per "appendere" del testo, ovvero inserire del testo nella posizione immediatamente successiva a quella del cursore
- *o* – per inserire del testo nella linea successiva a quella corrente

Dopo ognuno di questi comandi si entra nella modalità inserimento. Per uscire dalla modalità inserimento si preme Esc.

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)
- *a* – per "appendere" del testo, ovvero inserire del testo nella posizione immediatamente successiva a quella del cursore
- *o* – per inserire del testo nella linea successiva a quella corrente

Dopo ognuno di questi comandi si entra nella modalità inserimento. Per uscire dalla modalità inserimento si preme Esc.

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)
- *a* – per "appendere" del testo, ovvero inserire del testo nella posizione immediatamente successiva a quella del cursore
- *o* – per inserire del testo nella linea successiva a quella corrente

Dopo ognuno di questi comandi si entra nella modalità inserimento. Per uscire dalla modalità inserimento si preme Esc.

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)
- *a* – per "appendere" del testo, ovvero inserire del testo nella posizione immediatamente successiva a quella del cursore
- *o* – per inserire del testo nella linea successiva a quella corrente

Dopo ognuno di questi comandi si entra nella modalità inserimento. Per uscire dalla modalità inserimento si preme Esc.

Altri comandi bash e filtri

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 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 scrive su schermo
- *stderr (2)* – un canale per segnalare errori senza mischiarli con l'output, di default scrive su schermo

Con *pipe* e *redirezioni* si possono connettere in vari modi questi canali, creando delle vere e proprie catene 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 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 scrive su schermo
- *stderr (2)* – un canale per segnalare errori senza mischiarli con l'output, di default scrive su schermo

Con *pipe* e *redirezioni* si possono connettere in vari modi questi canali, creando delle vere e proprie catene 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 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 scrive su schermo
- *stderr (2)* – un canale per segnalare errori senza mischiarli con l'output, di default scrive su schermo

Con *pipe* e *redirezioni* si possono connettere in vari modi questi canali, creando delle vere e proprie catene 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 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 scrive su schermo
- *stderr (2)* – un canale per segnalare errori senza mischiarli con l'output, di default scrive su schermo

Con *pipe* e *redirezioni* si possono connettere in vari modi questi canali, creando delle vere e proprie catene 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 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 scrive su schermo
- *stderr (2)* – un canale per segnalare errori senza mischiarli con l'output, di default scrive su schermo

Con *pipe* e *redirezioni* si possono connettere in vari modi questi canali, creando delle vere e proprie catene 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.

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.

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.

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.

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.

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)

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)

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

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

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

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

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)

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 [2])
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.

[2] 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)

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)

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)

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)

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)

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)

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)

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*

Sostituzione di testo

```
echo night | sed s/night/day
```

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

awk

Nella versione più semplice, *awk* può dividere una serie di linee in formato tabulare ed effettuare delle operazioni su ogni elemento di ogni linea.

Elenco di tutti i programmi eseguiti da un utente

```
ps aux | awk '$1 == "pippo" {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

Awk (links)

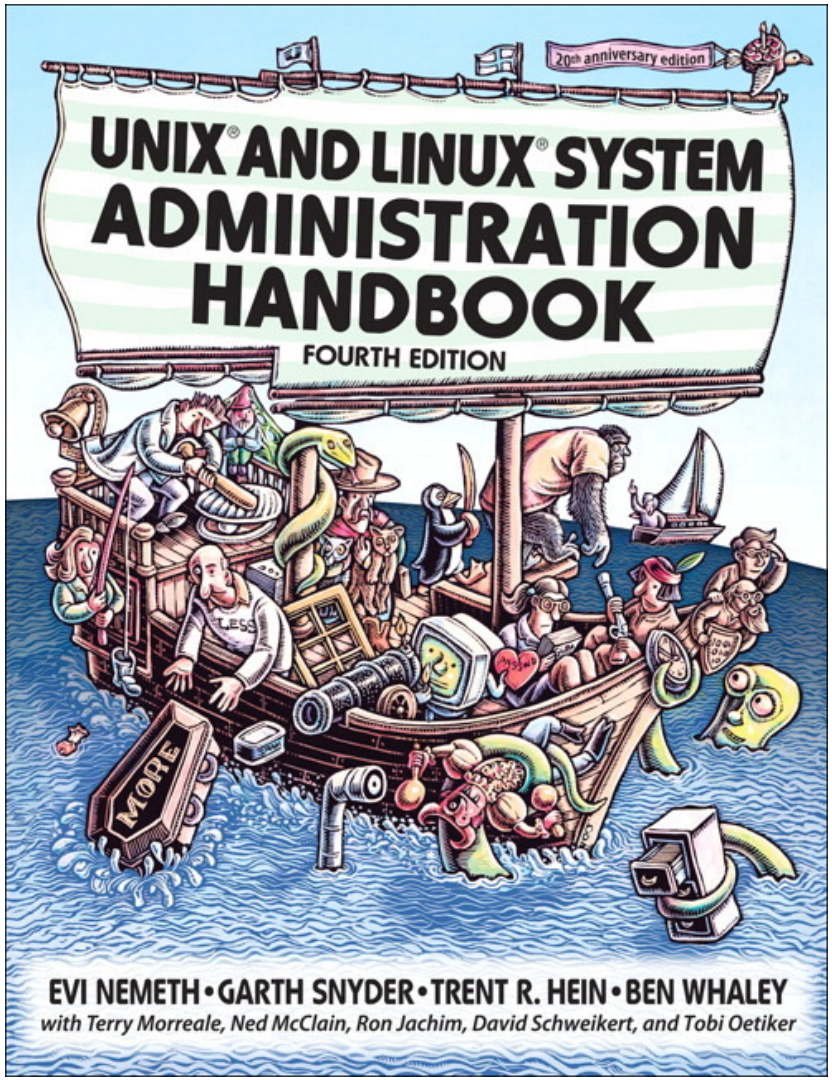
Effective AWK programming, il manuale della versione GNU di awk (quella distribuita con quasi tutte le distribuzioni di GNU/Linux):

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

Una buona guida ad awk:

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

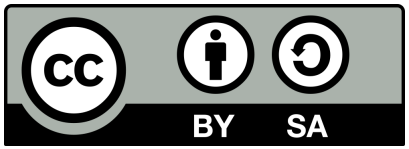
Per approfondire



Google is your friend

Goodbye and thanks for all the fish

Grazie per l'attenzione!



`https://www.poul.org`