Sistemi Operativi

AAF - Secondo anno - 3CFU

A.A. 2023-2024

Corso di Laurea in Matematica

Awk e altri comandi

Annalisa Massini

Dipartimento di Informatica
Sapienza Università di Roma

Argomenti trattati

1. Comandi su file di testo
   - Il comando `awk`
2. Comandi sui file
   - I permessi sui file

Altri comandi

Il comando awk

Comando awk

- L'utility awk serve per processare file di testo secondo un programma specificato dall'utente
- awk deriva dalle iniziali dei cognomi dei suoi autori, Alfred **A**ho, Peter **W**einberger e Brian **K**ernighan
- awk legge i file riga per riga ed esegue una o più **azioni** su tutte le linee che soddisfano certe **condizioni**
- Azioni e condizioni sono descritte tramite una sequenza che definisce un programma
- Nei moderni Linux, awk è un link a versioni più recenti come gawk e mawk

Comando awk

- La sintassi di awk è del tipo:
  `awk <opzioni...> 'codice awk' <file-di-testo>`
  - `codice awk` è un programma che specifica azioni e condizioni
  - oltre a comparire sulla linea di comando tra singoli apici `codice awk` può essere in un file (awk script) specificato con l'opzione `-f`
  - Ogni linea del file in input `file-di-testo` è vista come una sequenza di campi separati da tab e/o spazi
  - L'opzione `-F` serve per specificare un carattere separatore sostitutivo
- Più estesamente si ha: `awk [-F separatore] [-f file_awk] [-v var=var...] [programma_awk] [file...]`

Comando awk

- Un programma awk si può presentare come una lista del tipo:
  `[condizione1 [, condizione2, ...]] {azioni}`
- Per ogni riga del file in input, vengono valutate le condizioni e, se il risultato è vero, vengono eseguite i comandi del programma
- Per ogni riga possono essere eseguiti 0, 1 o più programmi
- Se ci sono più condizioni, separate da virgola, allora il programma viene applicato a tutte le righe che si trovano tra la prima riga che soddisfa la prima condizione e l'ultima riga che soddisfa l'ultima condizione

Comando awk

- Non mettere la condizione equivale a dire che il rispettivo programma va eseguito per tutte le righe
- Prima di essere eseguire condizioni e programmi, ogni riga del testo viene spezzata in campi (cioè ridotta in **token** o *tokenizzata*), usando un *field separator* `FS`
  - di default, `FS` è un qualunque spazio, ma con l'opzione `-F` lo si può ridefinire ad un qualunque carattere
  - lo stesso effetto lo si può ottenere assegnando un valore ad `FS` all'interno di un programma

Comando awk

- All'interno di condizioni e programmi si possono usare paramentri speciali, come ad esempio (ma non solo):

  `FNR` : numero di riga del file attuale
  `NR` : numero di riga tra tutti i file
  `ARGIND` : indice del file attuale (il primo ha indice 1)
  `NF` : numero di campi
  `FS` : separatore di campi
  `$1, $2, ...` : parametri posizionali per i campi della linea corrente (il numero di campo è tra 1 ed `NF`)
  `$0` : l'intera riga non spezzata

- si possono inoltre usare tutte le variabili eventualmente specificate con l'opzione `-v`

Comando awk

- Le **condizioni**, possono essere definite come segue:
  - espressioni contenenti operatori logici e/o relazionali
  - `(var)_o_(const) cmp (var)_o_(const)`: dove `cmp` è un operatore di confronto (`==`, `!=`, $>$, $<$, $>=$, $<=$)
  - le precedenti condizioni sono atomiche; possono essere combinate con AND (`&&`), OR (`||`) e NOT (`!`), e raggruppate con le normali parentesi
  - ci sono 2 condizioni speciali: `BEGIN` (vale solo prima della prima riga del primo file) e `END` (vale solo dopo l'ultima riga dell'ultimo file)

Comando awk

- Le **azioni** che costituiscono i programmi possono essere definite seguendo la sintassi del C/Java
  - assegnamenti con =
  - test di uguaglianza con ==
  - `for (init; cond; iter) istruzioni`
  - `while (cond) istruzioni`
  - `do istruzioni while (cond)`
  - `break` e `continue`
  - `if (cond) istruzioni`
  - `if (cond) istruzioni; else istruzioni`
  - se `istruzioni` è un blocco che contiene più istruzioni, va racchiuso dalle parentesi graffe
  - `exit` salta alla riga di comando successiva

Comando awk

Esempi per cominciare

- Creare un file amici.txt con linee composte da `nome, numero` (età, numero di telefono, data, ecc.)
  - Il comando `awk '/Anna/ { print $2 }' amici.txt`
    - `awk { print $n }` stampa il campo n di ogni linea di un file
    - `/nome/` seleziona tutte le linee che contengono nome
  - Scrivere `awk '($2>x) { print $2, $1 }' amici.txt`
    - Cosa viene visualizzato?
  - Scrivere `awk '($2>x && $2<y) { print NR, $2, $1 }' amici.txt`
    - Cosa viene visualizzato?

Comando awk

Esempi per cominciare

- Programma awk per sommare i valori del quinto campo di ogni record
  - `awk ' { totale += $5 } END { print "totale " totale " byte" }' file.txt`
- Programma awk che eseguito su `file.txt` estrae le righe pari oppure dispari
  - `awk '(NR % 2) { print }' file.txt`
  - `awk '((NR % 2) - 1) { print }' file.txt`
- Programma awk che eseguito su `file.txt` modifica il delimitatore nel contenuto del file (ad esempio nelle date del tipo gg/mm/aaaa produce gg-mm-aaaa)
  - `awk '$1=$1' FS="/" OFS="-" file.txt`

Comando awk

Esempi per cominciare

- Programma (*script*) awk che eseguito su `file.txt` stampa il contenuto del file tutto su di una riga:
  - `awk 'BEGIN {RS="\n"; ORS=" ";print "\n"} {print $0} END {print "\n \n"}' file.txt`
- Programma awk che eseguito su `file.txt` stampa il contenuto del file fino alla riga contenente la stringa **parola**
  - `awk '{print} /parola/ {exit}' file.txt`

Comando awk

- Alcune funzioni utilizzabili:
  - `length(s)`: ritorna la lunghezza della stringa `s` (il numero di elementi, se `s` è un array, ma non in tutte le versioni di `awk`)
  - `split(s, a, sep)`: tokenizza la stringa `s` nell'array `a` (distruggendolo se già esisteva; il primo indice è 1), usando il separatore `sep` (può non essere dato, e allora si usa `FS`); ritorna il numero di token ottenuti
  - `tolower(s)` e `toupper(s)`: ritornano la stringa `s` con le lettere tutte minuscole o tutte maiuscole
  - `int(d)`, `log(d)`, `exp(d)`, `sqrt(d)`, funzioni standard (la `int` non arrotonda, ma tronca)

Comando awk

Esempi per cominciare

- Programma awk che eseguito su file.txt stampa tutte le righe del file che superano la lunghezza di 40 caratteri
  - `awk 'length($0) > 40 { print $0 }' file.txt`
- Programma awk che eseguito su un file contenente nomi e date in formato gg/mm/aa consente di estrarre l'anno e il nome:
  - `awk '{ split($0,x,"/"); print x[3], $1 }' file-nomi`

Comandi sui file

I permessi sui file

I permessi dei file

- I permessi dei file servono a specificare chi può far cosa
  - **ogni file** ha associato un **utente** ed un **gruppo proprietari**
  - inizialmente, il proprietario è chi crea il file, ed il gruppo è il gruppo primario (ovvero, quello specificato per primo in `/etc/passwd`) di quell'utente
  - *inizialmente* perché si può usare il comando `chown` per cambiare il proprietario (vedere più sotto)
  - il proprietario decide cosa permettere e cosa no agli altri utenti e agli altri gruppi, definendo i **permessi** di file e directory

I permessi dei file

- Per i **file** i permessi sono: lettura `r`, scrittura `w` ed esecuzione `x`

Permesso	Ottale	Significato
- - -	0	Non si può fare niente (è però possibile vedere gli attributi, se i permessi sulla directory lo consentono)
- - x	1	Non si può fare niente (per eseguire, si deve prima leggere)
- w -	2	Si può scrivere, ma solo da riga di comando, sovrascrivendo completamente il file o appendendo dati alla fine (per altre modifiche, bisognerebbe prima leggere); si può anche cancellare, ma occorrono opportuni diritti sulla directory
- w x	3	Come il permesso 2
r - -	4	Si può leggere
r - x	5	Si può leggere ed eseguire
r w -	6	Si può leggere e modificare a piacimento (ma attenzione alla cancellazione)
r w x	7	Si può fare tutto (ma attenzione alla cancellazione)

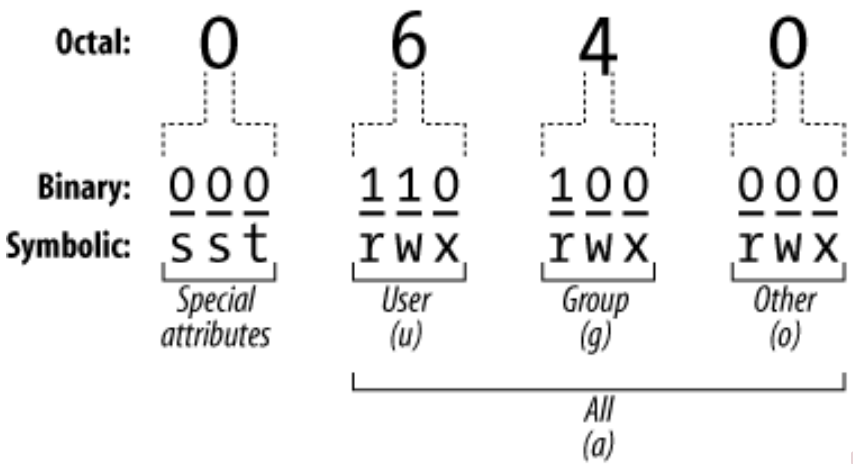
I permessi dei file

- Per le **directory** la cosa è un po' più complicata

Permesso	Ottale	Significato
- - -	0	Non si può fare niente
- - x	1	Si può settare come cwd (ma solo se c'è il permesso per *tutte* le directory nel path); si può anche *attraversare* (cioè vedere un file o una directory al suo interno, se c'è permesso in lettura)
- w -	2	Non si può fare niente (per fare veramente modifiche, occorrono i permessi di esecuzione)
- w x	3	Come il permesso 7, ma non si può listare il contenuto
r - -	4	Si può solo listarne il contenuto, senza vedere gli attributi dei file/directory contenuti (si può sapere se si tratta di file o di directory); non può essere *attraversata*
r - x	5	Si può leggere (attributi compresi), settare come cwd ed attraversare; non è possibile cancellare o aggiungere file/directory
rw -	6	Come il permesso 4 (write senza execute è inutile)
r w x	7	Si può fare tutto: listare contenuto, aggiungere file e directory, cancellare file contenuti (anche senza permesso di scrittura sul file!) e directory contenute (ma occorrono tutti i permessi)

I permessi dei file

- Per ogni file/directory, sono specificati 3 insiemi di permessi come quelli definiti sopra
  - il primo da sinistra è per l'**utente**: si applica se proprietario e utente utilizzatore coincidono
  - il secondo per il **gruppo**: si applica se l'utente utilizzatore appartiene al gruppo del file
  - il terzo per **tutti gli altri utenti**: si applica nei rimanenti casi
  - vengono mostrati da `ls -l` e `stat`
  - ci sono poi i permessi (o attributi) speciali



I permessi dei file

- Permessi speciali: **setuid bit** (s), **setgid bit** (s), **sticky bit** (t)
  - I permessi speciali consentono di impostare funzioni avanzate sui file o sulle directory
  - Sono rappresentati dai bit:
    - **setuid** - Set User Identification
    - **setgid** - Set Group ID
    - **sticky** - conosciuto come Sticky bit
  - I permessi speciali vanno usati con molta prudenza
  - Vengono visualizzati al posto del bit di esecuzione: il setuid nella terna utente, il setgid nella terna gruppo e lo sticky nella terna altro
  - Quando il permesso è impostato appare la lettera **s** per setuid bit e per setgid bit, mentre appare la lettera **t** per sticky bit: se è minuscola il permesso c'è, altrimenti è maiuscola (ed *inutile*)

I permessi dei file

- Il **setuid bit** si usa solo per i file **eseguibili**
- E' il permesso che consente l'esecuzione di un processo come proprietario del file e non come l'utente che ne richiede l'esecuzione
- Quindi impostando questa modalità su un file:
  - il file viene eseguito da qualunque utente del sistema con gli stessi privilegi dell'utente proprietario
  - i privilegi con cui opera il corrispondente processo sono quelli dell'utente proprietario del file e non sono quelli dell'utente che esegue il file,
  - ad esempio, se il proprietario è root, il file viene eseguito con i privilegi di root, indipendentemente da chi lo ha eseguito
  - Ad esempio, il comando `passwd` ha il setuid, che permette ad un utente di modificare la propria password

I permessi dei file

- Il **setgid bit** è l'analogo del *setuid bit*, ma vale per i gruppi
- I privilegi sono quelli del gruppo che è proprietario del file eseguibile
- Il **setgid bit** può essere applicato anche ad una directory e allora ogni file nella directory ha il gruppo della directory, anziché quello primario di chi crea files

- **Esercizio** provare a dare il comando `stat /tmp /usr/bin/passwd` e controllare cosa viene visualizzato

I permessi dei file

- Lo **sticky bit** inibisce il permesso di cancellazione dei file di un altro utente anche dove è autorizzata.
- Lo **sticky bit**, applicato su una directory, *corregge* il comportamento dei permessi write+execute (vedere Tabella directory): si possono cancellare file solo se si hanno i permessi di scrittura su quei file
- Per essere più precisi, lo sticky bit ha il seguente effetto:
  se una directory d appartiene all'utente u e un utente $u' \neq u$ cerca di cancellare un file f in d che non appartiene nè ad u' nè al gruppo cui appartiene u', allora
  senza sticky bit su d, per cancellare f è sufficiente avere i diritti di scrittura su d (nel gruppo *other*), anche se non si hanno i permessi di scrittura su f (sempre su *other*)
  con lo sticky bit per cancellare f sono necessari anche i permessi di scrittura su f

Il comando `chmod`

- Comando `chmod mode[, mode...] filename`: serve a cambiare i permessi
- Due modi di settare il *mode*: ottale o con lettere (simbolico)
  - **ottale**: si danno 4 numeri
  - Il primo numero indica setuid (4), setgid (2) e sticky (1), gli altri numeri vanno da 0 a 7, come nelle Tabelle, e sono per utente, gruppo ed altri
  - Si possono anche dare soltanto 3 numeri, e si intende che i bit speciali sono tutti a 0 (caso più comune)
  - **Esercizio** creare un file e settarne i permessi a `rws r-S -w-` e poi a `rwx r-- -wT` usando la modalità ottale

Il comando `chmod`

- Comando `chmod mode[, mode...]  filename`
  - **lettere**: qui se ne possono specificare molti, separati da virgole
  - il formato di ogni modo simbolico è :
    `[ugo][+-=][perms...]`
    dove `perms` è zero, una o più lettere nell'insieme {rwxst},
    oppure una lettera nell'insieme {ugo}
  - **Esercizio** creare un file e settarne i permessi a `rws r-S -w-` e poi a `rwx r-- -wT` usando la modalità simbolica

Il comando chmod: esempi ed esercizi

- **Esempio: Attivare suid**
  - Usando la modalità simbolica: `chmod u+s file.txt`
  - Usando la modalità ottale: `chmod 4755 file.txt`
  - Visualizzando si ha:
    ```
    -rwsr-xr-x 1 utente root 500 2020-10-03 18:46
    file.txt
    ```
- **Esempio: Disattivare suid**
  - Usando la modalità simbolica: `chmod u-s file.txt`
  - Usando la modalità ottale: `chmod 0755 file.txt`
  - Visualizzando si ha:
    ```
    -rwSr-xr-x 1 utente root 500 2020-10-03 18:46
    file.txt
    ```
- **Esercizio** togliere il permesso di lettura ad un file, e poi provare a visualizzarlo con `cat` o con `geany`
- **Esercizio** verificare che `chmod` modifica il ctime del file

Il comando `umask`

- Comando `umask [mode]`: è un comando della bash (non si trova in man, ma si può fare help umask)
  - Definisce quali permessi **negare** al momento della creazione di nuovi file e directory
  - Senza argomenti mostra l'umask corrente, altrimenti setta la maschera dei file al `mode` specificato
  - Si possono specificare solo le 3 terne, non i permessi speciali (che inizialmente sono tutti a 0)
  - I permessi predefiniti sono **666** per i file e **777** per le directory
  - Settando umask, alla creazione i permessi per un file saranno il risultato dell'operazione bit-a-bit *666 AND NOT(umask)*, mentre per una directory saranno *777 AND NOT(umask)*
  - `umask` ha effetto anche su `chmod`
  - **Esercizio:** modificare l'umask in modo che i permessi siano 664, sia che venga creata una directory che un file. Modificare poi in modo che il permesso per una nuova directory sia 775 e per un file sia 664.

I comandi `chown` e `chgrp`

- Comandi `chown [-R] proprietario {file}` e `chgrp [-R] gruppo {file}`: servono a cambiare proprietario o gruppo
  - Possono essere usati solo da root, quindi ci vuole `sudo` (altrimenti chiunque potrebbe creare un file con *contenuti compromettenti* e darlo ad utente ignaro)
  - Se si passano delle directory e c'è l'opzione `-R`, si cambiano ricorsivamente tutti i file e le directory in esse contenute
  - **Esercizio** riprendendo l'esempio dell'albero di directory `/home/utente1/dir1/dir3/dir7/`, creare un file (vuoto) `file`*i* dentro ciascuna directory `dir`*i*, e cambiare i proprietari in questo modo: la directory `dir3` diventa di `utente3`, tutti i file dentro `dir3` diventano di `utente2`, `dir7` diventa di `utente2` e `file7` diventa di `utente3`. Dopodiché, provare a cambiare i permessi di `file3` e `dir7`