Sistemi Operativi

AAF - Secondo anno - 3CFU

A.A. 2022-2023

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. Altri comandi
   - Altri comandi per elaborare contenuti di testo

Altri comandi

Il comando awk

Comando awk

- L'utility awk serve per processare file di testo secondo un programma specificato dall'utente
- L'utility 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 (la cui sintassi è simile al C)
- awk è il comando principe per elaborare contenuti di testo

Comando awk

- awk deriva dalle iniziali dei cognomi dei suoi autori, Alfred Aho, Peter Weinberger e Brian Kernighan
- Nei moderni Linux, awk è un link a gawk (GNU awk)
- In realtà awk è la *vecchia* versione, a descrizione seguente riguarda le versioni più recenti gawk e mawk

Comando awk

- `awk -Fc [-f filename] program {variable=value} {filename}*`
  - `program` è un programma che specifica azioni e condizioni
  - Tale programma può comparire sulla linea di comando tra singoli apici *oppure* in un file (awk script) specificato con l'opzione `-f`
  - Ogni linea del file in input è vista come una sequenza di campi separati da tab e/o spazi
  - L'opzione `-F` serve per specificare un carattere separatore c sostitutivo
- Più estesamente si ha: `awk [-F separatore] [--posix] [-f file_awk] [-v var=var...] [programma_awk] [file...]`

Comando awk

- Con awk si esegue un programma scritto in un linguaggio simile al C - però con accesso semplificato ai file (e alle loro righe) e senza necessità di compilare
- Quindi con awk si riesce a fare *tutto* su un file di testo
- awk lavora sull'input riga per riga
  - se l'input è da tastiera, dopo ogni pressione dell'invio awk valuta la riga e stampa eventualmente il suo output, quindi input ed output si vedranno alternati
  - se l'input dato dai file specificati come argomento, allora l'output non sarà intercalato all'input
- Quindi: awk specifica cosa occorre fare in una generica riga

Comando awk

- Più in dettaglio, un programma awk è una lista del tipo:
  `[condizione11 [, condizione12]] {programma1}`
  ⋮
  `[condizione`n`1 [, condizione`n`2]] {programma`n`}`
- Per ogni riga, vengono valutate le condizioni e, se il risultato è vero, viene eseguito il corrispondente programma
- Per ogni riga possono essere eseguiti 0, 1 o più programmi
- Se ci sono 2 condizioni sulla stessa riga, 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

- Il programma si può anche articolare su più righe
- Non mettere la condizione equivale a dire che il rispettivo programma va eseguito per tutte le righe
- Prima di essere passata a condizioni e programmi, ogni riga viene spezzata in svariati campi (o meglio, ridotta in **token**, cioè *tokenizzata*), a seconda del *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 alcune variabili speciali (**ce ne sono anche altre**, vedere il `man`):

  `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, ...` : campi della linea corrente, il generico campo n è compreso 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)
  - Ci sarebbe tutto il discorso sulle *espressioni regolari*, che però non trattiamo

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

- 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

- Alcune funzioni utilizzabili:
  - `printf` come nel C; quasi uguale al `PrintStream.print` del Java: scrive una stringa formattata come specificato
  - `gensub(s1, s2, n[, v])`: sostituisce, *ritornando il risultato*, tutte le occorrenze di `s1` con `s2` nella variabile `v` (se non data, `v` è `$0`)
  - `substr(s, da [, quanti])`: restituisce la sottostringa di `s` che inizia da `da` (il primo carattere è 1) ed è lunga `quanti` (se non dato, fino alla fine della stringa)
  - `index(s, t)`: restituisce il primo indice di `s` in cui comincia la sottostringa `t`, oppure 0 (quindi conta da 1...)

Comando awk

Esempi per cominciare

- Crea un file amici.txt con linee composte da `nome, numero` (di telefono, data, ecc)
  - digita `awk '/Anna/ { print $2 }' amici.txt`
  - `{ print $n }` stampa il campo n di ogni linea di un file
  - `/nome/` stampa tutte le linee che contengono nome
  - digita `awk '($2>`x`) { print $2, $1 }' amici.txt` per visualizzare...
  - digita `awk '($2>`x` && $2<`y`) { print NR, $2, $1 }' amici.txt` per visualizzare...

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 contenete la stringa **parola**
  - `awk '{print} /parola/ {exit}' file.txt`
- 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`

Comando awk

Esempi per cominciare

- Programma awk per sommare i valori del quinto campo di ogni record
  - `awk '1 { totale += $5 } END { print "totale:" totale "byte" }' 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`
- 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`

Altri comandi

Altri comandi per elaborare contenuti di testo

Comando sed

- Comando `sed [-e script] [-f file_script] [-r] [-s] [files...]`
  - versione semplificata di `awk`: tutto ciò che si può fare con `sed` si può fare anche con `awk` (il viceversa non vale), ma per alcune cose `sed` è più facile e conciso
  - vale quanto detto per `awk`, ma anziché programmi ci sono *azioni*, ovviamente più limitate dei programmi
  - inoltre, le singole righe non vengono tokenizzate e i files di input sono visti solo come una sequenza di righe
  - quindi uno *script* `sed` è fatto di coppie (condizione, azione)
  - lo *script* `sed` può essere fornito o direttamente tramite `-e` (o anche senza opzioni), o in un file a parte tramite `-f`

Comando sed

- Comando `sed [-e script] [-f file_script] [-r] [-s] [files...]`
  - Anche le condizioni sono più limitate di `awk`; alcuni esempi:

n	con $n \in \mathbb{N}$: vale vero alla riga n-esima
n, m	con $n, m \in \mathbb{N}$: vale vero dalla riga n-esima alla m-esima (incluse)
`/regex/`	vale vero se la riga *contiene* un match con la `regex`
`$`	vale vero all'ultima riga

  - Principale differenza con `awk`: tutte le righe che *non* soddisfano alcuna condizione vengono stampate in output così come sono (mentre sono ignorate da `awk`)
  - Altra grande differenza: se ci sono più condizioni vere, viene applicata solo la prima vera, **però** se, come risultato di un'azione, qualche condizione c successiva risulta vera, allora si applica anche l'azione corrispondente a c

Comando sed

Esempi

- `sed '4,$d' filename`
  - stampa a video soltanto le prime 3 righe del file `filename`, `d` è il comando di cancellazione che elimina dall'output tutte le righe a partire dalla quarta ($ sta per l'ultima riga del file)
- `sed 3q filename`
  - stesso effetto del precedente comando: in questo caso sed esce dopo aver elaborato la terza riga (3q)
- `sed /stringa/y/char1char2/new1new2/ filename`
  - sostituisce in tutte le righe che contengono la stringa `stringa` il carattere char1 con il carattere new1 ed il carattere char2 con il carattere new2
- `sed '/stringa/!y/char1char2/new1new2/' filename`
  - invece sostituisce... ma deve usare '

Comando sed

- Comando `sed [-e script] [-f file_script] [-r] [-s] [files...]`
  - per quanto riguarda le azioni, consideriamo le seguenti:

`r filename`	appende il contenuto di `filename` alla fine della riga
`d`	cancella la linea
`i \riga`	inserisce `riga` prima della riga
`a \riga`	inserisce `riga` alla fine della riga
`c \riga`	sostituisce `riga` alla riga

Comando sed

Esercizi

- `sed s/modello1/modello2` Sostituisce in ogni riga la prima occorrenza della stringa modello1 con la stringa modello2
  `sed s/modello1/modello2/g` Agisce su tutte le occorrenze di ogni riga di input controllata `/g` sta per globally
- **Esercizio**: provare i seguenti comandi su un file in input che contenga aaa e ao `sed s/a/A/g`
  `sed s/aaa/A/g`
  `sed s/'aaa'/A/g`
  `sed s/ao/A/g`
  `sed s/ao/AO/g`

Comando `tr`

- Comando `tr [-d] [-c] [-t] stringa1 [stringa2]`
  - Ancora meno di `sed`: si possono tradurre, eliminare e sostituire insiemi di caratteri
  - legge sempre da tastiera e scrive sempre su schermo
  - `stringa1` e `stringa2` sono successioni di caratteri,
  - in generale, `tr` rimpiazza ogni occorrenza del carattere *i*-esimo di `stringa1` con l'*i*-esimo di `stringa2`
  - con `-d` cancella tutti i caratteri in `stringa1` e `stringa2` non va data
  - con `-c`, si prendono i caratteri *non* presenti in `stringa1`; usata soprattutto in combinazione con `-d`
  - **Esercizio** vedere cosa succede se `stringa1` contiene caratteri ripetuti
  - **Esercizio** vedere cosa succede con l'opzione `-c`, soprattutto quando si traduce (quando si cancella è facile...)

Comando `uniq`

- Comando `uniq [-u] [-d] [-c] [filein [fileout]]`
  - elimina le righe identiche *consecutive*
  - l'input può essere da file o da tastiera; se viene dato il file di input, si può specificare un file di output (altrimenti, a schermo)
  - con `-c`, per ogni riga stampata dice anche quante volte era ripetuta
  - con `-d`, non stampa le righe singole (mai ripetute)
  - con `-u`, stampa solo le righe singole (mai ripetute)

Comando `sort`

- Comando `sort [-r] [-f] [-n] [-u] [-t sep] [-k POS1[,POS2]] [file...]`
  - ordinamento per righe dei file in input (o da tastiera)
  - ordinamento lessicografico: le cifre vengono prima delle minuscole, che vengono prima delle maiuscole (ma con `-f` non fa differenza tra maiuscole e minuscole); una parola p che è prefisso di una parola q è minore di q
  - se le righe contengono numeri e si vuole che `sort` li interpreti come tali, allora occorre usare l'opzione `-n`
  - ordinamento dal più piccolo al più grande; con `-r` dal più grande al più piccolo
  - con `-u`, stampa una sola volta le righe uguali
  - con `-k`, tokenizza ogni riga come fa `awk` (secondo gli spazi, o secondo il separatore specificato da `-t`), e poi ordina rispetto ai contenuti che vanno dal campo numero `POS1` al campo numero `POS2` (se `POS2` non è dato, allora va fino alla fine della riga)

Comandi `cut` e `wc`

- Comando `cut [-d sep] [-f fields] [file...]`
  - versione molto ridotta di `awk`: tokenizza ogni riga e ne stampa i campi dati
  - l'input può essere da file o da tastiera
  - `-d` serve per ridefinire il separatore
  - `-f` si aspetta una sequenza di range (separati da virgola)
- Comando `wc [-c] [-l] [-m] [-w] [-L] [file...]`
  - stampa statistiche su file di testo: numero di righe, di parole e di bytes
  - l'input può essere da file o da tastiera
  - le opzioni servono a controllare cosa stampare: `-c` numero di bytes, `-m` numero di caratteri (diverso dal precedente se ci sono caratteri accentati), `-l` numero di righe, `-w` numero di parole, `-L` numero caratteri della sola riga più lunga

Altri comandi

Comandi per avere informazioni

Comandi `id`, `who` e `uptime`

- Il comando `id [-u] [-g] [-G] [username]`: stampa user id, group id e tutti i gruppi cui l'utente `username` (o quello attuale, se `username` non è specificato) appartiene
  - `-u`: stampa solo lo user id
  - `-g`: stampa solo il group id
  - `-G`: stampa solo gli id dei gruppi cui `username` appartiene
- Il comando `who [-a]` specifica chi è attualmente loggato
  - mostra proprio tutti, comprese le schermate di testo attivabili con CTRL+ALT+F*n*
- Il comando `uptime` mostra da quanto tempo il sistema è stato avviato
  - con `-p` dà un output più comprensibile

Comandi `whoami` e `which`

- Il comando `whoami` stampa l'utente attualmente loggato
  - non sempre c'è il prompt con identificazione
  - non sempre ci si ricorda quale utente si sta impersonando
- Il comando `which [-a] comando` specifica dove si trova l'eseguibile relativo al comando
  - variabile d'ambiente `PATH`: contiene le directory dove cercare quando si dà un comando (scrivere `echo $PATH`)
  - un comando viene eseguito prendendo il primo eseguibile effettivamente trovato in una di quelle directory (ce ne potrebbe essere più d'uno, ma conta il primo match)
  - `which` mostra il path assoluto del file eseguibile relativo al comando
  - **Esercizio** provare `which ls`, `which -a ls` e `which -a which`
  - **Esercizio** provare anche `which cd`: i comandi built-in non hanno un file eseguibile, ci pensa direttamente bash