Sistemi Operativi

AAF - Secondo anno - 3CFU

A.A. 2020/2021

Corso di Laurea in Matematica

Altri comandi

Annalisa Massini

Dipartimento di Informatica
Sapienza Università di Roma

Argomenti trattati

1. Altri comandi
   - Comandi di visualizzazione
   - Comandi per confrontare file
   - Comandi per avere informazioni
   - Il comando `awk`
   - Altri comandi per elaborare contenuti di testo

Altri comandi

Comandi di visualizzazione

Comandi `less` e `more`

- Comandi `less {files}` e `more [-num] [+num] [-d] {files}`
  - come `cat`, ma paginano l'output se è troppo lungo
  - `less` è normalmente usato da `man` per mostrare il manuale
  - differiscono in svariati comportamenti:
    - `less` permette di muoversi sempre sia in avanti che all'indietro, `more` solo se usato senza pipelining (cioè il —)
    - si chiudono premendo `q`, ma per `less` sparisce tutto quello che era scritto, con `more` resta l'ultima schermata
    - `less` pagina sempre, `more` solo se l'output è più grande di una pagina
    - opzioni di `more`: `-num` setta a `num` il numero di righe in una pagina; `+num` comincia la visualizzazione dalla riga `num`, `-d` mostra una sorta di lista di comandi in basso

Comandi `head` e `tail`

- Comando `head [-c car] [-n righe] [file...]`
  - al solito, senza argomenti legge da tastiera (una riga per volta)
  - come `cat`, ma stampa solo i primi `car` caratteri o le prime `righe` righe (ciò che è minore) dei file dati
  - senza opzioni stampa 10 righe, senza limiti sui caratteri
  - **Esercizio** scrivere quanto segue su un file, e poi farsi stampare solo la prima riga, ma usando l'opzione `-c`:
    ```
    ciao
    addio
    ```
- Comando `tail [-n righe] [-f] [file...]`
  - al solito, senza argomenti legge da tastiera (ma questa volta occorre premere CTRL+d alla fine dell'input)
  - come `cat`, ma stampa solo le ultime `righe` righe dei file dati
  - con `-f`, aggiorna di continuo la stampa: utile se si vuole leggere un file cui vengono continuamente aggiunti dati (ad esempio, come risultato di una qualche computazione, *mentre* la computazione stessa è in esecuzione)

Comandi `head` e `tail`

- **Esercizio** trovare il modo per andare avanti ed indietro di k righe e di k pagine con `more` e `less`
- **Esercizio** creare un file con `gedit` (lanciato in background), scriverci dentro almeno 4-5 righe e poi salvare; tornare sulla shell ed eseguire `tail nomefile`. Poi aggiungere qualche altra riga, ed eseguire nuovamente `tail nomefile`. Effettuare nuovamente questi passaggi, ma eseguendo stavolta `tail -f nomefile`. È sufficiente scrivere le modifiche sul file, affinché vengano visualizzate dal `tail`? Perché?

Altri comandi

Comandi per confrontare file

Comandi `cmp` e `diff`

- Il comando `cmp [-b] file1 file2` confronta i 2 file dati
  - `cmp` si limita a trovare la prima occorrenza di un byte in cui differiscono
  - con `-b`, stampa anche tale byte, sia in ottale che nella versione stampabile, if any
- Il comando `diff [-i] [-b] [-r] [-q] file1 file2` elenca **tutte** le differenze
  - le differenze vengono mostrate riga per riga
  - `-b`: ignora le differenze, se consistono solo in un numero diverso di spazi
  - `-i`: ignora le differenze, se consistono solo nel *case* diverso (minuscolo vs. maiuscolo)
  - `-r`: confronta ricorsivamente 2 directory, confrontando con `diff` i file che hanno lo stesso percorso relativo e segnalando quali file sono presenti in una sola delle 2 directory
  - `-q`: dice solo se i file sono diversi, senza elencare le differenze

Comandi `cmp` e `diff`

- **Esercizio**
  - Creare un file e scriverci dentro qualche riga
  - Copiarlo in un altro file, farci delle modifiche e poi confrontare i 2 file sia con `diff` che con `cmp`
  - Rifare poi la stessa cosa, ma al posto della copia creare un link
  - Rifare la stessa cosa, ma con directory al posto dei file; modificare alcuni file e poi eseguire `diff` sia con che senza `-r`

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

Altri comandi

Il comando awk

Comando awk

- Il comando `awk [-F separatore] [--posix] [-f file_awk] [-v var=var...] [programma_awk] [file...]` chiama un programma per la manipolazione di dati di tipo testo
  - `awk` deriva dalle iniziali dei cognomi dei suoi autori, Alfred Aho, Peter Weinberger e Brian Kernighan
  - `awk` è il programma principe per elaborare contenuti di testo
  - In realtà `awk` è la *vecchia* versione, a descrizione seguente riguarda le versioni più recenti `gawk` e `mawk`
  - Questo programma è scritto in un linguaggio simile al C, dove però si semplifica l'accesso ai file (e alle loro righe) e non è necessario compilare
  - Quindi, si può fare praticamente *tutto*

Comando awk

- L'input di awk è dato dai file specificati come argomento
- Se non ci sono argomenti, awk legge da tastiera
- awk lavora sull'input riga per riga
  - se l'input è da tastiera, dopo ogni pressione dell'invio awk valuta la riga e stampa evantualmente il suo output, quindi input ed output si vedranno mischiati
  - se l'input è da file, allora l'output non sarà misto 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; 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
  `$`*n* : se *n* è compreso tra 1 ed `NF`, il valore dell'*n*-esimo campo
  `$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:
  - `(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 abbaimo trattato

Comando awk

- I **programmi** possono essere definiti 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

Comando awk

- Principali differenze con C/Java:
  - uso estremamente libero delle stringhe (lo ritroveremo negli script): la concatenazione tra variabili e/o costanti avviene senza operatori (tra costanti, lo fanno anche C e Java, ma con variabili no...)
  - confronto tra stringhe tramite ==
  - il ; è un separatore e non un terminatore, quindi può essere omesso dopo l'ultima istruzione
  - non serve dichiarare le variabili, siano esse semplici o array: ci si limita ad usarle
  - niente errore se una variabile viene usata prima che sia assegnata: varrà la stringa vuota "" (o anche zero, se viene usata come numero)

Comando awk

- Alcune funzioni utilizzabili:
  - `length(s)`: ritorna la lunghezza della stringa `s` (o se `s` è un array, il suo numero di elementi, 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
  - `int(d)`, `log(d)`, `exp(d)`, `sqrt(d)`, funzioni standard (la `int` non arrotonda, ma tronca)
  - `printf` come nel C; quasi uguale al `PrintStream.print` del Java: scrive una stringa formattata
  - `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 $2, $1 }' amici.txt` per visualizzare...

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 (più limitate dei programmi, ovviamente)
  - 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

  - grande 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

- 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

  - **Esercizio** scrivere uno script sed che, nelle righe che contengono solo numeri o spazi, sostituisca i numeri con la sola prima cifra del numero stesso; se una riga contiene un identificatore, allora eliminare da esso tutte le cifre; per tutte le altre righe, scrivere ciao sia prima che dopo la stringa

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]]`
  - 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)
  - elimina le righe identiche *consecutive*
  - 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, dove 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