Sistemi Operativi, Secondo Modulo, Canale M–Z
Riassunto della lezione del 16/03/2017

Igor Melatti

Comandi vari

- Comandi `less` {`files`} e `more [-num] [+num] [-d]` {`files`}
  - come `cat`, ma paginano l'output se è troppo lungo
  - nonostante i nomi, `more` è più limitato di `less`
  - `less` è normalmente usato da `man` per mostrare una pagina di manuale
  - differiscono in svariati comportamenti:
    * `less` permette di muoversi sempre sia in avanti che all'indietro, `more` solo se usato senza pipelining (ovvero, solo nel modo detto sopra)
    * tutti e due si chiudono premendo `q`, ma una volta chiuso `less` sparisce tutto quello che era scritto prima, con `more` resta l'ultima schermata
    * `less` pagina sempre, `more` solo se l'output è più grande di una pagina
    * far riferimento al `man` per una descrizione dei comandi interattivi ammessi da `more` e `less`
    * 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
    * tra i comandi interattivi c'è la ricerca di espressioni regolari: basta digitare `/`, seguita dall'espressione regolare (estesa); in `more` non trova le occorrenze nella pagina attuale, ma solo in quelle seguenti; in `less` trova tutte le occorrenze, evidenziandole
    * tra i comandi interattivi c'è la `h`, che mostra l'help dei comandi interattivi
    * **esercizio:** trovare il modo per andare avanti ed indietro di k righe e di k pagine con `more` e `less`
- Comando `man`, già visto in lezione 2

Sezione	Contenuto
1	User Command
2	System Calls
3	Library Functions
4	Special Files
5	File Formats
6	Games
7	Miscellaneous
8	System Administration and Privileged Commands
9	Kernel Interfaces (dipende dalla distribuzione)

Table 1: Sezioni del manuale

  - la lista completa delle sezioni è riportata in Tabella 1
  - **esercizio:** fare in modo che il contenuto di una pagina di manuale sia scritto tutto insieme, senza che occorra premere `q` per ritornare al prompt

- Comando `clear`
  - nessun argomento, nessuna opzione
  - pulisce lo schermo: ritorna il prompt in alto a sinistra

- Comando `echo [-n] [-e] stringa`
  - già visto, vengono ora precisati gli argomenti
  - stringa può contenere anche degli spazi
  - `-n`: non va a capo dopo aver stampato `stringa`
  - l'opzione `-e` abilita le sequenze di escape con il \: `\t` è la tabulazione, `\n` è l'andata a capo, `\xHH`, con `HH` cifre esadecimali, è il carattere con codice ASCII `HH` (vedere Figura 1: la prima colonna dà la prima cifra, la prima riga la seconda)
  - **esercizio:** tenendo presente la Figura 1, scrivere il carattere di tabulazione, seguito da una `c`, e andare poi a capo

- Comandi `cat [-T] [-n] [-E] [file...]` e `tac [file...]`
  - già visto, vengono ora precisati gli argomenti
  - `-T` anziché stampare le tabulazioni nel modo standard (ovvero aggiungendo spazi fino ad arrivare al prossimo multiplo di 7 nella riga corrente), stampa `^I`
  - `-n`: numera tutte le linee di output (e le precede con degli spazi)
  - `-E`: aggiunge $ alla fine di ogni riga
  - spiritosamente, `tac` stampa dall'ultima riga alla prima

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Figure 1: Il codice ASCII

  - usa come encoding quello mostrato in `echo $LANG`, che di solito è UTF-8 (che contiene il codice ASCII di Figura 1)
  - Quindi, ogni sequenza di byte letta dal file viene stampata seguendo lo standard dato

- Comando `od [-N bytes] [-j bytes] [-A base] [-t type] [-c] [file...]`
  - fa il dump di un file, ovvero ne mostra il contenuto in esadecimale
  - quindi, è possibile passargli anche file non di testo, come ad esempio gli eseguibili
  - infatti, qui non c'è nessuna interpretazione (come fa `cat`) dei bytes letti: viene stampato il loro valore come numero (decimale, ottale o esadecimale)
  - senza argomenti legge da tastiera, ma occorre specificare l'output per intero (premendo CTRL+d alla fine)
  - `-N bytes`: fai il dump di solo `bytes` bytes
  - `-j bytes`: fai il dump saltando i primi `bytes` bytes
  - `-t type`: scegli la base per il dump vero e proprio: `x` per esadecimale, `d` per decimale, `o` per ottale (default); si può anche specificare quanti bytes per ogni intero (solo potenze di 2, e fino ad un certo punto dipendente dall'implementazione); se si aggiunge una `z` al tipo stampa alla fine di ogni riga i caratteri stampabili
  - `-A base`: scegli la base per gli offset (prima colonna dell'output), stesse scelte del `-t` (non proprio vero, ma ok per i nostri scopi; vedere il `man`)
  - `-c`: mostra anche la corrispondenza con i caratteri stampabili

  - **esercizio:** farsi visualizzare i bytes che vanno dal decimo al trentesimo del contenuto dell'eseguibile del comando `more` (vedere anche il comando `which` più avanti), prima in ottale e poi in esadecimale, con la prima colonna sempre in esadecimale; se per caso qualche byte è un carattere stampabile, farselo stampare
  - **esercizio:** fare il dump di alcuni caratteri immessi da tastiera, e verificare con `echo` che la conversione sia esatta (sia in ottale che in esadecimale)

- 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)
  - **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?

- Comandi `cmp [-b] file1 file2` e `diff [-i] [-b] [-r] [-q] file1 file2`
  - confrontano i 2 file dati, ma esiste anche `diff3`
  - `cmp` si limita a trovare la prima occorrenza di un byte in cui differiscano (con `-b`, stampa anche tale byte, sia in esadecimale che nella versione stampabile, if any)

  - `diff`, invece, lista *tutte* le differenze; le differenze vengono mostrate riga per riga (2 righe che differiscono per un solo carattere verranno mostrate come differenti *tout-court*)
  - `-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 all'interno di tali directory, e segnalando quali file sono presenti in una sola delle 2 directory
  - `-q`: dice solo se sono uguali o se sono diversi, senza listare le differenze
  - **esercizio:** creare un file e scriverci dentro qualche riga; poi 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`

- Comando `patch [--dry-run] [-f] file diff_file`
  - si supponga di aver confrontato `file1` e `file2` con `diff`, e di aver salvato il risultato in `diff_file`
  - allora, il comando `patch file1 diff_file` trasforma `file1` in `file2`
  - con `--dry-run`, si limita a scrivere `file2` a schermo, senza modificare `file1`
  - con `-f`, non chiede nessuna conferma, come potrebbe accadere per alcune scelte
  - **esercizio:** creare un file e scriverci dentro qualche riga; poi copiarlo in un altro file, farci delle modifiche e poi confrontare i 2 file `diff`. Copiare il risultato dentro un nuovo file, e applicare la patch. Controllare con `diff` il risultato

- Comando `date [-s data] [-d data] [+formato]`
  - senza argomenti, scrive la data e l'ora (con fuso orario usato)
  - con `-s data`, cambia la data e la setta a `data` (solo con `sudo`)
  - con `-d data`, mostra la `data` (utile per fare conversioni)
  - con `+formato`, si cambia il modo con cui la data viene mostrata; `formato` può essere composto dalle seguenti sottoparti:
    - `%F`: scrive solo la data come YYYY-MM-DD; equivalente a `%Y-%m-%d`
    - `%H`: scrive solo l'ora (come in un orologio digitale)
    - `%M`: scrive solo i minuti (come in un orologio digitale)

    - `%S`: scrive solo i secondi (come in un orologio digitale, ma c'è anche il valore 60! Per farsi una cultura, cercare "leap second" o "secondo intercalare")
    - `%N`: scrive solo i nanosecondi
    - `%s`: numero di secondi intercorso dalla mezzanotte in punto del primo gennaio 1970 (più o meno l'accensione del primo Unix, detto anche *epoch*)
    - per gli altri, vedere `man date`
  - il formato di `data`, invece, è un po' più libero, vedere il `man` (ma anche `info date` per altre informazioni)
  - `-d` può servire a convertire da epoch a data umana: ad esempio il comando `date -d @1000 +%F-%H-%M-%S` mostra la data dopo 1000 secondi da epoch
  - **esercizio:** farsi stampare solo l'anno in cui ci troveremo, dopo 10 miliardi di secondi dal 1970

- Comando `find` {`dir`} {`espressione`}
  - altro comando con opzioni non semplici
  - serve per trovare file (che vengono cercati solo nelle directory `dir`), ed eventualmente effettuare delle azioni su di essi
  - le espressioni possono essere:

    opzioni le più importanti sono `-maxdepth M`, che impedisce di scendere per più di `M` sottodirectory in una delle directory di `dir`, e `-regextype T`, che permette di scegliere quali espressioni regolari usare: `T` può essere una tra `posix-awk`, `posix-egrep` e `posix-extended` (sono essenzialmente le ERE di lezione 5, con minime differenze; vedere `https://www.gnu.org/software/findutils/manual/html_node/find_html/posix_002dawk-regular-expression-syntax.html#posix_002dawk-regular-expression-syntax`) e `posix-basic` (le BRE di lezione 5)

    test se un test ritorna vero quando applicato ad un file, su tale file verranno applicate le azioni specificate; i test più importanti sono:
      - `-name pattern`: vero se `pattern` è soddisfatto; attenzione, `pattern` segue le regole del wildcarding, non delle BRE o delle ERE
      - `-iname pattern`: come `-name`, ma ignora la differenza maiuscole/minuscole
      - `-type [bcdolfls]`, dove `[bcdolfls]` è da intendersi come una BRE: ritorna vero se il file è del tipo specificato (ad esempio, `f` è per i file regolari, `l` per i link simbolici, `d` per le directory; vedere il `man`)

* `-size c[cwbkMG]`, dove `[cwbkMG]` è da intendersi come una BRE: ritorna vero se la dimensione del file è esattamente quella indicata (`c` sta per bytes, `k` per kB; vedere il `man`)
* `-user uname`: vero se il file appartiene all'utente `uname`
* `-perm mode`: vero se il file ha i permessi indicati in `mode`
* `-regex pattern`: come `-name`, ma questa volta `pattern` è interpretato come una BRE o come una ERE, dipendentemente da `-regextype`; inoltre, fa match con l'intero path ritornato (mentre `-name` fa match solo con il nome del file)
* `-atime T`: vero se il file è stato acceduto 24`T` ore fa; ci sono analogamente anche `-mtime` e `-ctime`
* `empty`: vero se il file (o la directory) è vuoto
* `-cnewer file`: vero se il file è più recente (come data di modifica) di `file`; ci sono analogamente anche `-anewer` e `-newer`
* in generale, si possono combinare tra di loro più condizioni di test, e verrà applicato l'AND logico tra di esse
* si possono anche usare NOT (`!`), AND (`-a`) ed OR (`-o`) tra le varie condizioni di test, con `\(\)` per raggruppare sottoespressioni

azioni da applicare ai file che superano i test; per essere più precisi, tali azioni vengono applicate, una alla volta, a ciascun file/directory che supera i test. Le azioni più importanti sono:

* `-ls`: applica `ls -dils` al file
* `-delete`: cancella il file
* `-exec command ;`: esegue il comando `command`, all'interno della quale si può usare {} al posto del nome del file
  - *sembra* facile ma... cominciano ad entrare in gioco le stranezze delle shell
  - inanzitutto, il `;`, dato così, verrebbe interpretato dalla shell stessa come separatore di comandi *prima* di essere passato come argomento a `find` (provare ad eseguire `echo ciao; echo ciao`)
  - quindi, occorre sostituirlo con uno dei seguenti: `\;`, `";"`, `';'` (provare ad eseguire lo stesso comando di cui sopra, sostituendo `;` con uno di questi 3 modi)
  - secondo, occorre che il `;` sia un argomento a parte di `-exec` (vedere com'è scritto nel `man`...); ovvero, `-exec` vuole 2 argomenti: il comando e il `;`
  - pertanto è necessario mettere (almeno) uno spazio prima del `;` "riscritto" (notare invece che scrivere `echo ciao;echo ciao` e `echo ciao ; echo ciao` è la stessa cosa)

∗ `-print`: stampa il nome del file (default action)

- ... e occhio a non confondere globbing con i pattern (soprattutto quelli con `-name`...); conviene sempre mettere i pattern tra `''`, così da evitare che la shell li interpreti prima di passarli a `find` (vedere esempio finale di lezione 5)
- **esercizio:** con una sola riga di comando, cancellare tutti i file, nel raggio di 3 sottodirectory da quella attuale, che abbiano un nome palindromo lungo 7 caratteri alfanumerici (è possibile usare `-name`?)
- **esercizio:** come prima, ma eseguire `ls` anziché cancellare, e limitarsi alla sola directory corrente. È proprio necessario usare `find`, o si può usare `ls`?
- **esercizio:** con una sola riga di comando, eseguire `ls` su tutti i file, nella directory attuale, che inizino con 2 caratteri alfanumerici, e poi continuino con stringhe non vuote qualsiasi. È proprio necessario usare `find`, o si può usare `ls`?
- **esercizio:** con una sola riga di comando, far sì che venga stampata una linea per ogni file che sia un link simbolico; tale linea dev'essere "Il file nomedelfile e' un link simbolico"

- Comando `id [-u] [-g] [-G] [username]`: stampa user id, group id, e i 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
- Comando `who [-a]` e `uptime`: informazioni sul sistema, ma disponibili a tutti
  - `who`: chi è attualmente loggato (con `-a` li mostra proprio tutti, comprese le schermate di testo attivabili con CTRL+ALT+Fn)
  - `uptime`: da quanto tempo il sistema è stato avviato
- Comando `whoami`: non sempre c'è il prompt come ce lo si aspetta, non sempre ci si ricorda quale utente si sta impersonando (magari dopo un logout...), quindi questo comando stampa l'utente attualmente loggato
- Comando `which [-a] comando`: 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

  - potrebbe succedere che ce ne sia più d'uno, ma per l'appunto conta il primo match
  - `which` mostra il path assoluto del file eseguibile relativo al comando
  - provare `which ls`, `which -a ls` e `which -a which`
  - provare anche `which cd`: i comandi built-in non hanno un file eseguibile, ci pensa direttamente bash

- Comando `time [-o file] [-a] [-f formato] [-v] [-p] comando`: raro caso di comando sia built-in che non
  - cioè esiste la versione built-in, che è quella che viene eseguita da Bash
  - per eseguire l'altra (che è più utile e ha tutte le opzioni riportate qui di seguito) occorre dare l'intero path (eseguire prima `which time`)
  - esegue `comando` (che può avere degli argomenti) e poi stampa statistiche sull'uso di CPU
  - con l'opzione `-o file` scrive il suo risultato (non quello di `comando`!) su `file`, sovrascrivendolo; se c'è anche l'opzione `-a`, appendendo alla fine
  - con l'opzione `-f formato` si può scegliere come dev'essere l'output di `time`; vedere `FORMATTING THE OUTPUT` nel `man time`
  - `-p`: usa il formato predefinito da POSIX
  - `-v`: dà tutte le informazioni, compreso l'uso della memoria
  - **esercizio:** confrontare le informazioni che `ps`, nel suo formato lungo con in più le informazioni su `vsz`, dà su se stesso con quello riportato da `time` nella sua versione completa