Il sistema Unix*
per l'utente VMS

Maurizio Loreti

Rev. 5
15 Ottobre 1992

1 Generalità

Entrati in ambiente Unix, ci si trova nell'interprete dei comandi (la *shell*); esistono varie shells, le più note sono la C-Shell (*csh*), la Bourne shell (*sh*) e la Korn shell (*ksh*): il default, settato dal System Manager (o meglio, per usare la terminologia Unix, dal *System Administrator*; il suo username è *root*) per ogni utente, è, almeno su MLIRIS, la C-shell, ed il sommario seguente si riferisce soltanto ad essa.

Il *file system* è analogo a quello del VMS, organizzato cioè ad albero in una catena di (sub)directories; unica differenza, il carattere di separazione tra le varie componenti del nome di un file non è più la parentesi quadra, ma la *slash* (**/**): ad esempio, **testi/unix**[1] indica il file di nome "unix" nella subdirectory "testi" della directory corrente (in VMS sarebbe stato **[.TESTI]UNIX**). La directory *radice* del disco è indicata da **/**, ed il nome di un file indirizzato in maniera assoluta, anziché relativa alla directory corrente, potrebbe essere qualcosa del genere di **/usr/cdf/loreti/testi/unix**.

La directory *radice* di ogni utente (in VMS: **SYS$LOGIN**) è indicata dalla tilde (il simbolo **~**): **~/testi/unix** si riferisce al file "unix" nella subdirectory "testi" della mia radice, e **~castro/dati/top** ad un file nell'albero delle subdirectories dell'utente avente username "castro".

Il punto (**.**) individua la directory *attuale*, e se ripetuto (**..**) la directory immediatamente superiore: **../a/filename** è l'equivalente Unix del VMS **[-.A]FILENAME**.

Attenzione: Unix è *case sensitive*; "file", "FILE" e "File" individuano tre files differenti. Anche le nostre password e tutti i comandi sono case sensitives, e se vengono dati con maiuscole e minuscole in forma sbagliata non verrano riconosciuti.

2 I comandi principali

I comandi alla shell hanno tutti un formato del tipo:

```
COMANDO [FLAG [FLAG ... ]] [ARGOMENTO [ARGOMENTO ... ]]
```

e se sono troppo lunghi possono essere conclusi da una backslash (****), la quale informa la shell che il comando continuerà sulla riga seguente (equivalente del **-** VMS).

I *flags* (o *switches*) sono l'equivalente dei qualifiers VMS, e modificano il significato del comando; sono generalmente, per distinguerli dagli *argomenti*, preceduti dal segno meno (il carattere **-**), così come nel VMS sono preceduti da una **/**). Per esempio, l'equivalente del **DIR** del VMS è il comando **ls** (list); tra i possibili flags c'è **-l** (long), e **ls -l** è

* A grandi linee...

[1] Nel corso del testo, frasi che compaiono **con questi caratteri** indicano comandi per il computer, o comunque terminologia relativa al sistema Unix (od al sistema VMS).

l'equivalente Unix di **DIR/FULL**. Più flags per lo stesso comando possono essere date sia separate che in successione, cioè sono forme valide ed equivalenti sia **ls -a -l** che **ls -al**.

Ad ogni sessione Unix sono associati tre *streams* di caratteri: lo standard input (*stdin*), lo standard output (*stdout*) e lo standard error (*stderr*); esattamente come **SYS$INPUT**, **SYS$OUTPUT** e **SYS$ERROR** sotto VMS. Normalmente sono assegnati tutti al terminale; ma molti comandi accettano l'input da stdin e/o producono l'output su stdout, e se o l'input o l'output devono essere invece su file, questo deve essere specificato nel comando stesso in una forma che è detta *redirection*.

Facciamo un esempio concreto: il comando **cat** (concatenate) accetta come argomenti i nomi di uno o più files, che scrive in successione su stdout: cioè, dando il comando **cat a b c**, il contenuto dei files a, b e c viene stampato (in quest'ordine) sullo schermo. Se invece l'intenzione è quella di creare un file su disco che contenga a, b e c (comando VMS **APPEND**) il comando va scritto nella forma **cat >newfile a b c**. L'operatore "**>**" informa appunto la shell che lo standard output deve essere *ridiretto* sul file newfile; attenzione, *se newfile esisteva già, verrà ricoperto e andrà perso*. Esiste anche una forma alternativa, "**>>**", che crea newfile se questo non esiste, o gli *appende* quello che viene da stdout se newfile esisteva già.

Per ridirigere lo standard input l'operatore è "**<**". Ad esempio, per spedire messaggi ad altri utenti si usa il comando **mail user1 user2**; **mail** legge da stdin un messaggio concluso da un end-of-file (CTRL-D in Unix), che sarà poi mandato agli utenti "user1" e "user2"; se si vuole inviare un messaggio che è già pronto su disco in un file di nome "mess", basta ridirigere lo standard input dando il comando nella forma **mail <mess user1 user2**.

Un comando utile per visualizzare il contenuto di un file è **more** (l'equivalente VMS è **TYPE/PAGE**): **more** accetta come argomento (opzionale) una lista di nomi di files, che vengono scritti su stdout una pagina alla volta (per vedere la pagina successiva di testo bisogna premere la barra degli spazi); e, se non è dato nessun argomento, l'input viene preso da stdin. Con **more** è conveniente usare l'ultimo degli operatori di redirezione della shell, "**|**" (detto *pipe*), il cui effetto è quello di dirottare lo stream di output di un programma dentro lo stream di input di un altro programma: con un esempio concreto, se si vuole vedere comodamente il contenuto di una directory nella quale ci sono più files di quanti stiano su uno schermo si può usare un comando **ls -l dirname | more**. L'effetto è quello di un **DIR/FULL** della directory "dirname" il cui output, invece di essere presentato direttamente sullo schermo, viene dato in pasto a **more** che ce lo farà vedere una pagina alla volta.

In tabella 1 sono riportati alcuni dei comandi VMS più comuni, ed i loro equivalenti Unix; di alcuni di questi si è già parlato: ecco ora brevi cenni sugli altri. **cp** copia files: la sintassi è **cp input output**, dove "output" può essere tanto il nome di un file quanto quello di una directory (però già esistente): **cp old new** copia il file "old" nel file "new", **cp file dirname** copia "file" nella directory "dirname" (in VMS sarebbe: **COPY FILE [.DIRNAME]***). E' possibile usare delle *wildcards* come nel VMS: l'asterisco (*****) ha lo stesso significato, per cui per copiare tutti i files dalla directory "thisdir" in quella corrente il comando sarebbe **cp thisdir/* .**. Il comando **mv** richiede gli stessi argomenti di **cp**, ed è equivalente ad un copy seguito poi da un delete dei vecchi files; e può servire anche al posto del VMS **RENAME**.

man comando è l'help in linea per "comando"; se non si ha idea di quale sia il comando opportuno in una certa situazione, basta dare il flag **-k** seguito da una *keyword*: ad esempio **man -k password** presenta su stdout una lista di tutti i comandi collegati alla parola chiave "password", con una brevissima descrizione per ognuno.

Altri comandi che possono essere utili sono: **sort**, che mette in ordine alfabetico leggendo da stdin e scrivendo su stdout; **head -N**, che scrive le prime N linee di un file su stdout; **tail -N**, che scrive le ultime N righe di un file sempre su stdout. Se ad esempio si vogliono conoscere i 5 files più grossi della directory attuale, il comando da dare

Comandi più usati		
VMS	UNIX (C-shell)	Mnemonica
date,time	date	
write sys$output	echo	
dir	ls	list
dir/full	ls -l	list long
show default	pwd	print working dir.
set def [a.b.c]	cd /a/b/c	change directory
set def [.a.b]	cd a/b	
set def [-]	cd ..	
set def sys$login	cd	
type	cat	concatenate
type/page	more	
delete	rm	remove
delete/confirm	rm -i	
copy	cp	copy
rename	mv	move
create/dir	mkdir	make directory
delete/dir	rmdir	remove directory
help	man	manual
@commandfile	source commandfile	
! commento	# commento	

Tabella 1: Raffronto tra i comandi più comuni in Unix e VMS

è **`ls -s | sort -n -r | head -5`**. La prima parte crea la lista della directory col flag **`-s`** (**`/SIZE`**), scrivendo quindi all'inizio di ogni riga, prima del nome del file, la sua dimensione in blocchi; la seconda ordina la lista numericamente (**`-n`**) in ordine decrescente (**`-r`**); la terza stampa le prime cinque righe del risultato finale.

Un altro comando utile è **`grep pattern file [file ... ]`**, che stampa su stdout tutte le linee dei files richiesti nelle quali compaia la pattern data come primo argomento (in VMS: **`SEARCH`**).

3 Simboli, alias ed environment variables

L'equivalente dei *simboli* VMS si chiama in Unix un *alias*: ad esempio, il comando **`alias dir 'ls -al'`** definisce il simbolo "dir" come equivalente alla string **`ls -al`**; da notare che, visto che la string conteneva uno spazio, la si è dovuta racchiudere tra due apici. **`alias`** dato da solo, senza argomenti, lista su stdout tutti gli alias definiti; **`unalias`** distrugge poi un alias esistente.

Ogni volta che viene creata una shell viene eseguito il file *.cshrc* della directory radice dell'utente; in pratica questo avviene sia quando si fa login sulla macchina e si entra nella propria shell, sia quando si chiede l'esecuzione di una procedura di comandi contenuti in un file, per la quale viene creata una *sub-shell*. All'atto poi del login soltanto, *dopo* .cshrc viene eseguito anche il file *.login*; quindi .cshrc è candidato a contenere, per esempio, le definizioni degli alias di uso generale che presumibilmente useremo anche nelle nostre procedure di comandi, mentre in .login si mettono comandi da eseguire una sola volta all'inizio, come la stampa di data e ora o messaggi di benvenuto.

Files il cui nome comincia con un punto sono speciali: non vengono listati da **`ls`**

(bisogna usare il flag **-a**, all, per vederli), e non vengono matchati[2] dalla wildcard, per cui **rm *** distrugge in realtà tutti i files della directory corrente meno quelli che cominciano con un punto. Questi due files, .cshrc e .login, devono essere ovviamente modificati a piacere da ogni utente.

Altre abbreviazioni utili della shell sono: **!$** (che sta per *l'ultimo* argomento del comando precedente), e **!*** (che sta per *tutti* gli argomenti del comando precedente). Ad esempio, se per listare il file z.c si è appena dato il comando **cat z.c**, volendolo compilare, nel comando successivo basta scrivere **cc !$**. Un altro uso possibile è: **alias dir 'ls -al \!* | more'**; dopo aver definito questo alias, ad esempio, **dir dir1 dir2** verrà tradotto nell'equivalente **ls -al dir1 dir2 | more**. Da notare l'uso dell'*escape character* ****: se esso non fosse stato presente, la sequenza **!*** sarebbe stata sostituita dalla shell (nel corpo del comando alias) dagli argomenti dell'ultimo comando prima eseguito; così, invece, viene passata al comando alias senza che il **!** venga interpretato dalla shell.

I *nomi logici* del VMS sotto Unix si chiamano *environment variables*; quelli di uso generale sono di solito definiti, per ogni utente, nel suo file .cshrc, attraverso il comando **set name = value** (**set** è l'equivalente del VMS **DEFINE**), od il comando equivalente **setenv name value**. Due di queste environment variables sono particolarmente importanti: quella che specifica il tipo di terminale che stiamo adoperando (*term*) e quella che specifica quali sono le directories in cui guardare per cercare i files eseguibili richiamati dai nostri comandi (*path*). Se vogliamo sapere quale è il path di default fissato dal system administrator, basta dare il comando **echo $path** (**echo** scrive su stdout i suoi argomenti, ed il dollaro sostituisce alla variabile il suo significato); e se vogliamo aggiungere alla serie di directories che costituisce questo default quelle che contengono i nostri comandi, supponiamo ad esempio che si chiamino **~/bin** e **~/paw**, basta inserire in .cshrc uno statement del tipo **set path = ($path ~/bin ~/paw)**. E' da notare sia l'uso delle parentesi tonde, sia come in questo modo le nostre directories compaiano nel path *dopo* quelle di sistema; se avessimo voluto che comparissero *prima*, ad esempio per usare una nostra versione di qualche comando di sistema come **ls** invece di quella normale, avremmo dovuto scrivere invece **set path = (~/bin ~/paw $path)**. Per listare tutti i nomi logici basta dare **set** senza argomenti; e per distruggere un nome logico c'è il comando **unset** (VMS **DEASSIGN**); e, per quel che riguarda il terminale, in genere basta un **set term = vt100**.

4 Protezioni

Ogni file ha dei codici di protezione (in Unix *file modes*), visibili con il comando **ls -l**; all'estrema sinistra della lista, da sinistra verso destra, ci sono dieci caratteri il cui significato è: tipo del file (il primo): **d** per directory, **-** per file normale; poi, tre codici di accesso per il propietario (*user*), tre per il *gruppo*, e tre per l'utente generico (*other*), che possono essere a loro volta **r** (*read*, permesso di lettura); **w** (*write*, permesso di scrittura, e anche di distruggere) o **x** (*execute*, permesso di esecuzione). Se ad esempio la string dei codici è del tipo **-rwxr-x--x**, vuol dire che si tratta di un file normale, per cui l'user ha diritto di lettura, scrittura ed esecuzione; il gruppo diritto di lettura ed esecuzione; e gli others solo diritto di esecuzione. Per cambiare le protezioni si usa il comando **chmod** (change mode): **chmod o+w file** dà diritto di scrittura (**+w**) agli others (**o**); **chmod go-x file** toglie diritto di esecuzione (**-x**) al gruppo (**g**) ed agli others (**o**); e così via. Se non sono specificati user, gruppo e/o others, la modifica è su tutti e tre i campi: **chmod -w file** toglie diritto di scrittura all'user, al gruppo e agli others.

[2] Che brutta parola!

5 Procedure di comandi

Un file di comandi è un file che contiene istruzioni alla shell, e viene eseguito col comando **source** seguito dal nome del file; parametri opzionali possono essere dati alla procedura mettendoli, come in VMS, dopo il nome del file da eseguire (se alcuni di essi contengono spazi vanno racchiusi tra apici). A questi parametri nel corpo della procedura ci si riferisce usando i simboli **$1** (il primo), **$2** (il secondo) e così via, proprio come in VMS dove i simboli analoghi sono **P1**, **P2** e così via). Se si pensa che il file di comandi possa essere richiamato anche da shell diverse dalla C-shell (per le quali i comandi sono differenti), la *prima* riga deve essere *obbligatoriamente* costituita dalla string (sintatticamente un commento) **#!/bin/csh**: essa informa il sistema che questa è una procedura di comandi, e che deve essere eseguita dal programma **/bin/csh**—cioè appunto dalla C-shell. La shell ha poi costrutti opportuni per tests condizionali (if-then-else) e loops da inserire nel file di comandi. Se il file che contiene i comandi viene reso eseguibile (**chmod +x filename**) può poi essere richiamato in esecuzione dal solo nome.

6 Foreground e background

Comandi la cui esecuzione si prevede di lunga durata possono essere eseguiti in *background*; se il programma **mc** è ad esempio un montecarlo, il comando **mc** lo eseguirà in interattivo (*foreground*); mentre **mc &**, con un **&** come ultimo carattere della linea, lo esegue in background, restituendo subito il prompt della shell dopo aver stampato una linea del tipo **[N] xxx**. **N** è il numero progressivo del processo in background: 1 per il primo, 2 per il secondo e così via; e quando il processo di background numero N termina, la shell scrive su stdout **[N] - Done**, informandone così l'utente. Se si vuole interrompere il processo N che sta venendo esguito in background prima della fine, si da o il comando **stop %N** (sospende il processo; si possono eseguire dei comandi; poi il processo stesso può essere fatto riprendere ancora in background col comando **bg** o passandolo in foreground col comando **fg**); o quello **kill %N** (lo termina del tutto). Se più di un processo in background è sospeso, bisogna specificare quale riprendere, dando il comando nella forma **bg %N** o **fg %N**.

Un programma che sta venendo eseguito correntemente in foreground può essere interrotto con un CTRL-Z; dopo aver eventualmente eseguito altri comandi dalla shell, lo si può rimandare in esecuzione ancora in foreground con **fg**, o passarlo in background con **bg**. La lista di tutti i lavori in background si ha col comando **jobs**; per la lista di tutti i lavori in background fatti partire anche da altre shell o altre sessioni, il comando è **ps**. Un processo in background *non può richiedere input dal terminale*.

Per uccidere del tutto un programma che sta venendo eseguito in foreground bisogna dare CTRL-C; se, infine, all'atto del logout ci sono dei processi (in foreground o in background) che sono stati sospesi, la shell ce ne informa.

7 La history

Tutti i comandi dati in precedenza alla shell si possono vedere listati su stdout dando il comando **history**; in questa lista ognuno è preceduto da un numero progressivo. Dando alla shell il comando **!N**, quello dei comandi precedenti che nella lista della history è associato al numero N viene richiamato e rieseguito tale e quale (cioè senza che sia possibile editare la command string); un'altra maniera di ottenere lo stesso risultato è quella di riferirsi ai comandi precedenti non mediante la numerazione progressiva ma mediante una string caratteristica: **!ls** riesegue l'ultimo comando che *cominciava* con i caratteri ls, **!?myfile?** riesegue l'ultimo comando in cui *compariva* la string "myfile".

I riferimenti all'ultimo comando dato sono collegati ad una sintassi più semplice: per rieseguire l'ultimo comando basta dare la sequenza **!!**; e per rieseguire l'ultimo comando però sostituendovi al posto della string "s1" l'altra string "s2" la sequenza è **^s1^s2**.

Per richiamare uno dei comandi diverso dall'ultimo aggiungendovi alcuni caratteri alla fine, ci si serve di uno dei modi su esposti di riferimento ai comandi precedenti, seguito dalle aggiunte: per esempio **!13 abc** manda in esecuzione il comando numero 13, al cui testo originale vengono aggiunti uno spazio ed i caratteri "abc"; lo stesso effetto hanno comandi del tipo **!ls abc** o **!?myfile? abc**. Invece **!{13}abc** aggiunge alla fine del comando numero 13 i caratteri "abc" senza alcuno spazio interposto; in mancanza delle parentesi graffe, il meccanismo della history avrebbe invece interpretato il tutto come un riferimento all'ultimo comando iniziante con i caratteri "13abc".

Per sostituire in un comando precedente alcuni caratteri con un'altra sequenza (ad esempio la string "s1" con quella "s2"), la sintassi da usare è invece **!13:s/s1/s2/**; oppure **!ls:s/s1/s2/**; o ancora, infine, **!?myfile?:s/s1/s2/**. In questi tre esempi al posto del carattere **/** si può usare un qualunque carattere non alfanumerico che non compaia né in s1 né in s2; ed inoltre la sostituzione viene eseguita solo *sulla prima* substring s1 incontrata nel comando richiamato. Per la sostituzione di *tutte* le sequenze s1 con s2 bisogna scrivere **:gs/s1/s2/**, in cui il **g** sta per global.

8 Editing

Esiste su MLIRIS una versione Unix dell'editor *edt*; per usarlo, bisogna mettere nel path la directory **~loreti/edt/bin** e definire il nome logico **BBC** attraverso un **setenv BBC ~loreti/edt/lib**; inoltre il nome logico **term**, che identifica il tipo di terminale usato, deve essere stato definito al login (attraverso **set term = ...**) come uno di quelli che compaiono nella directory **~loreti/edt/lib/term**. Se esiste nella propria directory radice un file di nome *edtini.edt*, esso viene eseguito da **edt** appena questi viene fatto partire.

9 Programmi FORTRAN

Il compilatore FORTRAN si richiama in esecuzione con il comando **f77**. Per compilare un file sorgente di nome "prova.f" (**.f** è il *file type*, o *estensione*, default per i sorgenti FORTRAN), il comando è **f77 prova.f** (attenzione, il **.f** va specificato esplicitamente); **f77** crea direttamente l'eseguibile (il **.EXE** del VMS) nel file **a.out**. Per compilare il file sorgente prova.f e creare un eseguibile di nome "prova", il comando da usare è **f77 -o prova prova.f**.

Un programma può essere costituito da più routines suddivise in vari files; in tal caso ognuno di questi va compilato separatamente per produrre un file oggetto (avente estensione **.o**, l'analogo dei **.OBJ** VMS), e poi tutti questi files oggetto vanno linkati assieme alle librerie FORTRAN per produrre l'eseguibile. In tal caso la sequenza di comandi da dare è: **f77 -c prova1.f** (che crea il file oggetto prova1.o a partire da prova1.f); analogamente si compilano tutti gli altri files sorgente; e poi il comando **f77 -o prova prova1.o prova2.o ...** esegue il link di tutti gli oggetti per creare l'eseguibile **prova** (al solito le estensioni **.o** vanno specificate esplicitamente).

Una libreria di routines può essere creata con il comando **ar** (archive): più esattamente, **ar r libp.a p1.o p2.o p3.o** crea una libreria "libp.a" contenente le routines dei file oggetto p1.o, p2.o e p3.o (o, se la libreria già esiste, vi aggiunge, o sostituisce, queste stesse routines). Le librerie hanno in genere estensione **.a** (archivio), ed *è opportuno che inizino* con le tre lettere *lib*. Per istruire il linker all'uso di una particolare libreria, ad esempio "libp.a", il comando da usare è **f77 -o prova prova.o -lp**: **f77** premette automaticamente le tre lettere **lib** ed aggiunge l'estensione **.a** alla **p** che segue immediatamente il

flag **-l** (libreria). Le librerie vengono cercate, di default, nella directory **/lib**; se si trovano in altre directories, bisogna che queste siano esplicitamente dichiarate con il flag **-L**. Ad esempio, nel comando **f77 -o prova prova.o -L/usr/cern/lib -lpawlib** il flag **-L** specifica che tutte le librerie successivamente citate nel comando devono essere cercate in **/usr/cern/lib**, e quello **-l** richiede di usare **libpawlib.a** per il link.

L'equivalente Unix dell'*unità logica NN* è il file **fort.NN** (così come sotto VMS è il file **FOR0NN.DAT**); per far leggere da un file su disco i dati che si riferiscono all'unità logica NN, bisogna creare un *soft link* tra il nome **fort.NN** ed il nome reale del file, con l'istruzione **ln -s filename fort.NN**. Attenzione: il soft link *viene scritto* nella directory corrente, per cui sia *non vale al di fuori di essa*, sia anche *sopravvive al logout*; per rimuoverlo bisogna dare esplicitamente un **rm fort.NN** (ossia lo si cancella come un file reale). Le unità 5 e 6 sono già automaticamente connesse a stdin e stdout, per cui operazioni di lettura e scrittura relative a queste unità avvengono da terminale e possono essere ridirette con le solite opzioni della shell.

Se un programma è molto grosso, la sua dimensione si può ridurre con il comando **strip progname**, che rimuove dall'eseguibile la symbol table del loader ed altre informazioni ridondanti. *Attenzione*: è da fare solo quando un programma è ben debuggato, altrimenti esso non è in grado di generare messaggi di errore comprensibili.

Se un programma gira su VAX ma non sulla Silicon Graphics, provate prima a compilare con **f77 -static ...**; poi, a controllare che tutte le variabili siano state inizializzate prima di essere usate; infine, se le variabili **REAL** sono usate come **REAL*4** (normali) o **REAL*8** (double precision); anche scrivere 1.0 al posto di 1.0D0 può creare problemi!

Per default il compilatore FORTRAN usa un livello minimo di ottimizzazione (**-O1**); per usare una ottimizzazione più spinta si possono dare i flags **-O2** o **-O3** (quest'ultimo solo se il comando **f77** genera l'eseguibile direttamente dai sorgenti); si guadagna facilmente un 30-50% sul tempo di CPU, ma la compilazione ed il linking sono molto più lunghi, per cui è meglio ottimizzare in questo modo solo programmi già ben debuggati. Altri flags utili sono **-w0** (produce messaggi di avvertimento su variabili non inizializzate o non usate); e **-check_bounds** (controlla che al run-time gli indici degli arrays siano compresi entro i limiti dichiarati).

Si può ottenere infine il flowtrace di un programma, per studiare quali sono le routines critiche in cui va speso il tempo di CPU:

```
f77 -p -o prova prova.f    ! Crea "prova" da "prova.f" con il flag
                           ! "p" (profiling); viene creato un file
                           ! di nome "mon.out" con le informazioni
prova                      ! relative. Si esegue poi il programma.
prof prova mon.out >p.out  ! Si esegue il flowtracer ("prof") su
                           ! prova, sfruttando le informazioni di
                           ! mon.out e mandando le informazioni
more p.out                 ! elaborate sul file p.out; questo
                           ! viene esaminato sul terminale
rm p.out                   ! ("more") e infine cancellato ("rm").
```