Amministrazione di Reti di Calcolatori L-A

Fondamenti per l'utilizzo dell'ambiente Unix ed approfondimenti sulla gestione delle risorse locali

L'interfaccia testo

Login

Riga di comando

Shell: fondamenti

Shell: scripting

Job control

L'interfaccia testo

Lo strumento più potente,flessibile e soprattutto più standard con cui amministrare un sistema Unix è l'interfaccia testuale a riga di comando.

Componente essenziale è la *shell* o interprete dei comandi, che permette di svolgere compiti quali job control, mette a disposizione variabili e strutture di controllo per scrivere semplici programmi, e permette di invocare gli eseguibili installati nel sistema.

Una delle shell più usate è la *bash*, un'evoluzione della classica Bourne Shell di Unix.

Login

Per accedere alla shell è necessario effettuare il login, ovvero indicare il proprio nome utente e provare la propria identità per mezzo, comunemente, di una password.

La shell

Una volta ottenuto, per mezzo del login, l'accesso al sistema, l'utente ha a disposizione una interfaccia per presentare le proprie richieste al sistema e per visualizzare le relative risposte.

L'interfaccia si occupa di interpretare, mediante una opportuna sintassi, i comandi dell'utente trasformandoli in richieste specifiche al sistema operativo.

In unix/Linux e' possibile selezionare diversi tipi di interfaccia utente, sia per l'accesso testuale (shell) che per l'accesso grafico (desktop environment).

L'utente 'root'

La maggior parte delle operazioni normalmente eseguite sulla macchina non richiede particolari privilegi. Un utente standard puo' tranquillamente utilizzare la macchina per programmare, editare testi, collegarsi ad internet…

Le operazioni di manutenzione, di installazione e di configurazione dei pacchetti devono invece essere eseguite da un utente particolare (**root**) che possieda opportuni diritti di scrittura e modifica delle directory e dei file 'critici'.

E' sempre buona norma, anche per gli utenti che conoscono la password di root, utilizzare un account standard per le operazioni di routine.

Shutdown

Anche lo spegnimento di una macchina Linux (e unix in generale) richiede alcune operazioni.

Si può abbandonare semplicemente la sessione di lavoro con exit (dalla shell di login) o con logout. Il sistema rimane comunue attivo e pronto ad altri accessi.

Lo spegnimento richiede invece, di norma, l'accesso di root. Alcune installazioni permettono a chiunque si trovi in possesso della console (tastiera) di eseguire lo spegnimento (shutdown) premendo ALT-CTRL-CANC.

Shutdown

root puo' invocare direttamente lo spegnimento, tipicamente con il comando:

shutdown [-h|-r] now

-h indica la richiesta di arrestare il sistema (esiste anche il comando **halt**),
-r indica la richiesta di riavviare il sistema (esiste anche il comando **reboot**).

now indica quando eseguire l'operazione. E' possibile, ad esempio, lasciare agli utenti collegati alcuni minuti per terminare il proprio lavoro.

Cambiare identità

L'utilità di questi comandi diviene ancora più evidente se si considera che un utente puo' modificare durante la sessione di lavoro la propria "identità" con il comando 'su':

su tipicamente (senza argomenti) consente l'accesso come root ma può anche consentire l'accesso come altri utenti:

su alex

naturalmente l'operazione richiede di conoscere la password dell'utente nel quale ci si vuole "trasformare", a meno che a lanciare il comando non sia root.

Chi sono, chi c'e'

Poiché è del tutto comune disporre di differenti identificativi utente con cui lavorare, è utile disporre di un comando per sapere quale e' l'identificativo con il quale si sta operando:

whoami	indica il proprio username
id	dà informazioni sull'identità e sul gruppo di appartenenza
who	indica chi e' attualmente collegato alla macchina

La command line

La shell indica il proprio stato di 'pronto' con una stringa di caratteri visualizzati nella parte iniziale della prima linea vuota. Questa stringa e' detta 'prompt'.

I caratteri digitati dall'utente dopo il prompt e terminati dal fine linea (ritorno a capo) costituiscono la command line.

Questa, tipicamente, contiene comandi e argomenti.
I comandi digitati sulla command line possono essere:
- built-in
- comandi esterni

La command line

Un built-in e' un comando direttamente eseguito dal codice interno della shell che lo interpreta e lo 'converte' in azioni sul sistema operativo. Un esempio tipico è costituito dai comandi per la navigazione del filesystem.

Un comando esterno e' un file eseguibile che viene localizzato e messo in esecuzione dalla shell (tipicamente in un processo-figlio). La shell può attenderne o meno la conclusione prima di accettare nuovi comandi. Il risultato di un comando esterno è un codice di ritorno ed un file (standard output).

Gli argomenti

Ogni comando, sia built-in che esterno, può accedere ai caratteri che seguono la propria invocazione sulla command line. I gruppi di caratteri, separati da spazi, rappresentano gli argomenti, cioè i dati su cui si vuole che il comando operi.

Un argomento che inizia con il carattere '-' e' indicato come opzione, e normalmente non è un vero e proprio dato da elaborare ma un modo di chiedere al comando di comportarsi secondo una certa modalità. Più opzioni possono essere solitamente raggruppate in un'unica stringa.

Gli argomenti

Esempi:

ls /home	arg #1="/home" indica la dir di partenza per la lista
ls -l /home	arg #1=opzione **l** indica che si desidera un listato in forma lunga
ls -l -a /home/alex	arg #1 e #2 = opzioni l (lunga) ed a (all)
ls -la /home/alex	arg #1 = opzioni concatenate l+a (identico a prima)

Man pages

Una installazione standard di unix mette a disposizione innumerevoli pagine di manuale raggruppate in sezioni:

(1) User commands
(2) Chiamate al sistema operativo
(3) Funzioni di libreria, ed in particolare
(4) File speciali (/dev/*)
(5) Formati dei file, dei protocolli, e delle relative strutture C
(6) Giochi
(7) Varie: macro, header, filesystem, concetti generali
(8) Comandi di amministrazione riservati a *root*
(n) Comandi predefiniti del linguaggio Tcl/Tk

Man pages

L'accesso alle pagine si ottiene con il comando
man <nome della pagina>

Spesso il nome della pagina coincide con il comando o il nome del file di configurazione che essa documenta.

Alcune opzioni utili sono qui riassunte:

man –a <comando>	cercherà in tutte le sezioni
man <sez.> <comando>	cercherà nella sezione specificata
man –k <keyword>	cercherà tutte le pagine attinenti alla parola chiave specificata

***Per avere altre informazioni sul comando* man è *ovviamente sufficiente usare* man man.**

Elencare i file

ls elenca i file o il contenuto della directory specificati come argomento; senza argomenti elenca il contenuto della directory corrente. Le opzioni più comuni sono:

-l abbina al nome le informazioni associate al file
-a non nasconde i nomi dei file che iniziano con .
-A come -a ma esclude i file particolari '.' e '..'
-F postpone il carattere '*' agli eseguibili e '/' ai direttori
-d lista il nome delle directory senza listarne il contenuto
-R percorre ricorsivamente la gerarchia
-i indica gli i-number dei file oltre al loro nome
-r inverte l'ordine dell'elenco
-t lista i file in ordine di data/ora di modifica (dal più recente)

Shell expansion

La bash permette di specificare nomi di file per mezzo di *pattern* (schemi). Quando si scrive una linea di comando contenente uno schema, la bash la interpreta in 2 passi:

1. Sostituisce lo schema con l'elenco di tutti i nomi di file che si adattano a tale schema
2. Esegue la riga di comando risultante

Shell expansion

Gli schemi vengono composti usando caratteri speciali:

***** rappresenta una qualunque stringa di zero o più caratteri
? rappresenta un qualunque carattere singolo

[$c_1c_2c_3$] rappresenta un qualunque carattere purchè appartenente all'insieme. Anche **range** di valori: **[c_0-c_n]**
Es. ***ls [q-s]**** lista i file con nomi che iniziano con almeno un carattere compreso tra q e s

[!$c_1c_2c_3$] rappresenta un qualunque carattere purchè non appartenente all'insieme
Es. ***ls *[!0-9]*** lista i nomi dei file che terminano con caratteri non numerici

Esempi d'uso delle wildcard

ls * lista i file del direttorio corrente (nel caso vi siano direttori cosa succede?)

ls [a-p,1-7]*[cfd]?
lista i file i cui nomi hanno come iniziale un carattere compreso tra 'a e 'p' e tra 1 e 7
Il penultimo carattere deve essere c, f, oppure d.

echo * esegue l'echo del carattere '*', privato del suo significato

ls *[!*\?]* lista tutti i file del direttorio corrente che non contengono una wildcard * o ?

Esempi d'uso delle wildcard

ls /*/*/* lista tutti i file dei direttori di secondo livello a partire dalla root

ls -d /*/*/* lista tutti i file dei direttori di secondo livello a partire dalla root (i direttori sono trattati come file)

ls [a-z]*[0-9]*[A-Z] lista i file i cui nomi iniziano con una minuscola, terminano con una maiuscola e contengono almeno un carattere numerico

ls *[!0-9]* lista i file del direttorio corrente i cui nomi non contengono alcun carattere numerico

Esempi d'uso delle wildcard

ls [a-z,A-Z,0-9][a-zA-Z0-9] lista i file con nomi di due caratteri alfanumerici (entrando nei direttori)

echo [a-z,A-Z,0-9][a-zA-Z0-9] lista i file con nomi di due caratteri alfanumerici

Altri comandi di uso generale sul filesystem

df visualizza lo spazio utilizzato e disponibile su ogni filesystem
du visualizza l'uso di spazio di un file o una directory
rm cancella un file o, meglio, rimuove il link
cp copia un file o piu' file in una directory
mv sposta un file o piu' file in una directory
ln crea un link ad un file
mkdir crea una directory
rmdir cancella una directory
chown cambia il proprietario di un file
chgrp cambia il gruppo proprietario di un file

Navigazione nel filesystem

Ogni processo, shell compresa, possiede un'informazione di stato che indica il nome della directory corrente nel file system. Normalmente, all'atto del login, ogni utente si trova nella propria home directory.

In ogni istante si può conoscere la directory corrente con il comando **pwd** (print working directory); è possibile modificare la directory corrente spostandosi cosi' nel file system utilizzando il comando built-in **cd**.

- cd invocato senza argomenti posiziona l'utente nella propria home
- cd seguito da un argomento posiziona l'utente nella directory con tale nome.

Navigazione nel filesystem

Si ricordi che ogni directory di unix contiene due directory speciali:
. rappresenta la directory stessa
.. rappresenta la directory superiore (tranne nella radice */*)
e che ogni percorso che inizia con la barra viene considerato assoluto, cioè relativo alla radice, mentre se inizia con qualsiasi altro carattere viene considerato relativo alla directory corrente.

Es. di spostamento assoluto
cd /home/alex
Es. di spostamento relativo
cd ../pippo/pluto

Variabili

Le variabili sono un modo offerto dalla shell per memorizzare delle stringhe di testo sotto un dato nome. La modifica o la creazione di una variabile si ottengono semplicemente indicando sulla command line il nome della variabile seguito da **=** e dal valore che le si vuole attribuire.
Nota: non inserire spazi prima e dopo il simbolo **=**

pippo=valore assegna "valore" alla variabile pippo

Il meccanismo di shell expansion si applica anche agli elementi della riga di comando che iniziano con **$**: $pippo viene sostituito con il valore assegnato alla variabile pippo

Comando echo

Per visualizzare un messaggio, unix mette a diposizione il comando echo che stampa i caratteri che lo seguono. Questo è immediatamente utilizzabile per visualizzare il valore di una variabile:

echo PATH visualizza "PATH"
echo $PATH visualizza il contenuto della variabile PATH
echo $pippo visualizza "valore"

PATH=.:$PATH modifica la var PATH inserendo la dir corrente

Variabili di ambiente

Vi sono alcuni dati, solitamente riguardanti il sistema o le preferenze di un utente, che sono utili a tutti i comandi (es. la versione del sistema operativo, la lingua dell'utente, ecc…). Sarebbe faticoso e ripetitivo doverli passare come argomento ad ogni comando che si esegue: per questo unix dispone del meccanismo delle variabili di ambiente.
La shell distingue le variabili d'ambiente dalle variabili standard (che invece rimangono confinate alla shell stessa) per mezzo dell'*esportazione*.

export pippo

Variabili di ambiente

La variabile esportata è *copiata* ai comandi discendenti dalla shell, non *condivisa*: non è quindi possibile per un processo discendente modificare le variabili di ambiente del padre.

Le operazioni di export e di assegnamento possono essere contemporanee:
export pluto=valore

esempio:

man ls=> output in inglese
export LANG=it
man ls => output in italiano

Quoting

Il meccanismo di espansione di wildcard e variabili è potente ma interferisce con l'interpretazione letterale di alcuni simboli: i già visti **[ ] ! * ? $** ed anche **{ } ( ) " ' ` \ | > < ;**

Quando si debbano passare come parametro ad un comando delle stringhe contenenti tali simboli, è necessario proteggerli dall'espansione.

**** non interpretare il carattere successivo come speciale

Es. ***ls ***** lista i nomi dei file che contengono il carattere * in qualunque posizione

Quoting

' (virgolette singole) ogni carattere di una stringa racchiusa tra virgolette singole viene protetto dall'espansione e trattato letteralmente, senza eccezioni.

" (virgolette doppie) ogni carattere di una stringa racchiusa tra virgolette doppie viene protetto dall'espansione, con l'eccezione del **$**, del backtick (`), di ****, ed altri casi particolari

Ricerca dei comandi

Tra le variabili d'ambiente comuni, la shell utilizza PATH per eseguire la ricerca dei comandi nel file system. La sua struttura è quella di un elenco di directory separate da **:**

PATH=/bin:/usr/bin:/sbin

Si noti che se si vuole mettere in esecuzione un comando (o programma) che si trova nella directory corrente (ad esempio, un compilato) la variabile PATH deve contenere la directory **.** Questa non è una buona norma, perché è facile lanciare per distrazione comandi errati.

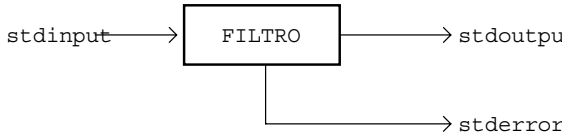
Ricerca dei comandi

È comunque sempre possibile lanciare un comando specificandone il percosro esplicito in modo relativo o assoluto, anche se al di fuori di PATH:
/usr/local/bin/top
./mycommand

In un sistema possono essere presenti diverse versioni di uno stesso comando in diverse directory. Il sistema quando non si usa un percorso esplicito lancia quella che trova nella directory che appare per prima in PATH.
which Permette di sapere quale versione si sta usando:
which passwd
/usr/bin/passwd

Stream predefiniti

Si è detto che in unix qualsiasi cosa viene astratta come file. In particolare, qualsiasi comando unix ha a disposizione 3 file con cui comunicare con il resto del sistema:
standard input in ingresso
standard output in uscita
standard error in uscita



Stream predefiniti

Per i comandi lanciati da un terminale, lo standard input viene agganciato alla tastiera e gli altri due al video

Esempi:

Il comando cat riporta sullo standard output i caratteri letti dai file che vengono nominati come argomenti:

cat prova.c hello.c
cat senza argomenti preleva i caratteri dallo standard input e li riversa sullo standard output

Concatenazione di comandi

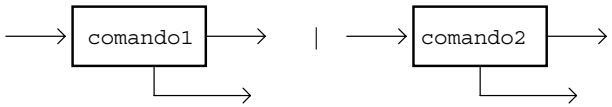
Il comando **less** riversa i caratteri provenienti dal suo standard input allo standard output arrestando l'operazione al raggiungimento della pagina (cioè dopo aver riempito un numero di righe pari all'altezza del terminale).

Il comando less, si comporta come un *filtro* che apporta modifiche al flusso di caratteri in ingresso per produrre un flusso di caratteri in uscita. In unix esistono diversi comandi che si comportano come filtri e la shell mette a disposizione un metodo per il collegamento dei vari flussi di caratteri: la **pipe**

L'indicazione dei collegamenti avviene sulla command line, intercalando ogni comando al successivo con il simbolo **|**

Concatenazione di comandi

Ad esempio, se si vuole utilizzare il filtro less sull'output del comando cat:
cat /etc/passwd | less



All'atto pratico la concatenazione fa apparire due comandi come uno solo, avente per STDIN quello del primo, STDOUT quello del secondo, e SDTERR la sovrapposizione dei due.
E' possibile comporre filtri comunque complessi concatenando più filtri fra loro:
ls | grep ^pro.a$ | less

Funzionamento della pipe

La pipe e' un oggetto messo a disposizione dal sistema operativo unix per collegare i flussi di dati di due processi. Il sistema si occupa della gestione e della sincronizzazione di questo sistema di comunicazione che rappresenta uno dei cosiddetti sistemi IPC.
A differenza della pipe fornita dal DOS, quella di unix non fa uso di file temporanei ma si appoggia interamente alle capacita' di buffering del sistema operativo che provvede a 'bloccare' il processo produttore di dati quando il processo ricevitore risulti piu' lento e, alternativamente, bloccare il processo ricevitore quando il produttore risulti piu' lento.

Funzionamento della pipe

Entrambi i processi, comunque, sono contemporaneamente attivi senza presentare la latenza tipica dei sistemi DOS. Questo permette la concatenazione di più comandi senza che l'output venga rinviato al completamento dell'ultimo.

Ovviamente, se un comando della pipeline deve operare delle modifiche su tutto lo stream di dati (es. ordinamento), allora l'output sarà disponibile solo al termine della lettura.

La pipe si puo' considerare una forma particolare di *ridirezione*. Il meccanismo della pipe esegue un collegamento (ridirezione) fra lo stdout del comando a sinistra della pipe con lo stdin del comando alla destra.

Ridirezione dell'output su file

E' possibile riversare i dati prodotti da un comando (ad esempio la lista prodotta da ls) su un file, utilizzando i simboli di ridirezione **>** e **>>**.

ls > miofile mette il risultato di ls nel file miofile, senza presentare nulla a video.

ls >> miofile esattamente come sopra, con la differenza che, se miofile esiste, allora l'output di ls viene aggiunto al termine del contenuto corrente

Ridirezione dell'error su file

Se si vogliono ridirigere solo i messaggi di errore si deve utilizzare il simbolo **2>** o **2>>**:

ls nomedifileinesistente 2> miofile

Lo stderr puo' anche essere ridiretto all'attuale stdout, con il simbolismo:

ls > miofile 2>&1

ridirige l'output di ls a miofile ed i messaggi di errore di ls all'attuale stdout (miofile).

Ridirezione dell'input da file

E' possibile anche utilizzare un file come sorgente di dati riversandone il contenuto sullo stdin di un comando:

more < miofile utilizza il contenuto di miofile come inpout al comando more.

Gestione dei processi

Ogni comando lanciato da shell o dal sistema diviene un ***processo***. I processi sono identificati univocamente da un numero chiamato Process ID (PID)

Sorvegliare i processi e poterne bloccare, ripristinare, o alterare l'esecuzione (ad esempio alzandone o abbassandone la priorità) è uno dei compiti fondamentali dell'amministrazione del sistema.

Un processo svolge le proprie azioni a nome dell'utente che lo ha lanciato (i processi lanciati da root hanno il potere di assumere l'identità di altri utenti, così facendo si "declassano" e perdono il potere di tornare indietro)

Gestione dei processi

Il comando **ps** è il modo più semplice di ottenere l'elenco dei processi attivi.

I parametri più utili di **ps** sono:

ax visualizza tutti i processi, anche non propri
u mostra gli utenti proprietari del processo
w visualizza la riga di comando completa che ha originato il processo
f visualizza i rapporti di discendenza tra processi

Segnali

Uno dei meccanismi con cui i processi possono comunicare tra loro e con il s.o. è quello dei segnali.

Ogni processo può "registrare" presso il sistema operativo una routine di gestione (**handler**) per un segnale. Quando un altro processo invia il corrispondente segnale, il processo a cui è destinato viene sospeso e viene eseguito l'handler.

Diversi tipi di segnali hanno caratteristiche quali:
- comportamento predefinito in assenza di handler
- possibilità di essere ignorati
- comportamento imposto

Segnali

I segnali possono essere inviati dal s.o. per indicare situazioni d'errore o comunque degne di attenzione, ad esempio:

7	SIGBUS	Bus error
8	SIGFPE	Floating Point Exception
11	SIGSEGV	Segmentation Violation
13	SIGPIPE	Broken Pipe
14	SIGALRM	Alarm expired
17	SIGCHLD	Child ended

Segnali

… oppure possono essere originati da altri processi per comunicare al destinatario una richiesta (solo il proprietario di un processo e root possono inviargli segnali), ad esempio:

1	SIGHUP	Hangup
9	SIGKILL	Kill
15	SIGTERM	Terminate
18	SIGCONT	Continue if stopped
19	SIGSTOP	Stop process

Segnali

Normalmente l'invocazione di un comando da shell blocca la shell per tutto il tempo di esecuzione. In tale intervallo la shell intercetta la pressione di alcune combinazioni di tasti e invia al processo un segnale:

Ctrl + Z → SIGSTOP
Ctrl + C → SIGKILL

Per inviare esplicitamente un segnale ad un processo si usa il comando **kill** con primo argomento il segnale e secondo il PID

kill –SIGKILL 150

Job control

Si può usare un'unica shell per l'esecuzione contemporanea di più comandi che non abbiano necessità di accedere al terminale, lanciandoli in background (sullo sfondo).

Questo si ottiene postponendo il carattere **&** alla command line. La shell risponde comunicando un numero tra parentesi quadre (job id) che identifica il job localmente a questa shell.

Se si lancia una command line senza **&**, e si vuole rimediare, si può dare un segnale di **STOP** con Ctrl+Z. Anche in questo caso si riceve un job id. Con il comando **bg %***job_id*, si invia un segnale CONT che riavvia il processo e contemporaneamente lo si mette in background.

Job control

Un processo in background non riceve più comandi dal terminale, poiché la tastiera torna ad agire sulla shell; se è necessario riportare in foreground (primo piano) un processo ricollegandolo così al terminale, si usa il comando **fg %***job_id*.

Il comando **jobs** mostra l'elenco dei job, cioè di tutti i processi avviati dalla shell corrente, indicando il loro stato (attivo o stoppato).

nohup

Ogni comando lanciato in una shell da luogo ad un processo che, normalmente, viene terminato dall'invio del segnale SIGHUP al termine della sessione di lavoro (chiusura della shell). Se si desidera creare un processo che 'sopravviva' al termine della sessione (es. un download lungo) bisogna renderlo immune dall'hangup.

Questo si ottiene anteponendo il comando **nohup** che provvede, inoltre, a scollegare l'output del processo dal terminale se non fatto esplicitamente nell'invocazione.

Di default, nohup dirige l'output sul file 'nohup.out'

Alias

La shell mette a disposizione alcuni sistemi per memorizzare linee di comando complesse in comandi più semplici da invocare. Uno di questi è l'**alias** che permette di attribuire un nome ad una command line:

alias miols='ls -l'

Le associazioni definite con alias, così come la definizione di qualunque variabile, vanno perse al termine della sessione.

File di configurazione della shell

Se si desidera ottenere un alias o una variabile persistenti, è possibile inserirne le definizioni in uno dei file di configurazione della shell, che vengono riletti ad ogni esecuzione della shell stessa. Questi file sono molteplici:

1) se la bash è attivata in seguito ad un login:
   prima **/etc/profile**
   poi il primo accessibile tra **~/.bash_profile**, **~/.bash_login**, **~/.profile**
   ed all'uscita **~/.bash_logout**
2) se la bash è invocata in altro modo:
   ~/.bashrc

nota: il carattere **~** è speciale e viene sostituito dalla bash con la homedir dell'utente.

Script

I file di configurazione costituiscono un esempio di uno strumento shell più generale: lo script. Lo script è un elenco di comandi che possono essere interpretati dalla shell, cioè una combinazione qualunque (purchè di sintassi corretta) di comandi esterni, built-in, e operazioni su variabili.

Molte attività di configurazione e gestione di un sistema unix/Linux vengono svolte per mezzo di shell script come, ad esempio, la procedura di avvio dei servizi al boot.

Script

Uno shell script è un file di testo con flag di eseguibilità impostato e, opzionalmente ma caldamente consigliato, che inizia con la stringa #!/bin/sh o #!/bin/bash.

Tale stringa viene vista dalla shell stessa come un commento (come ogni stringa preceduta da **#**), mentre per il s.o. #! rappresenta il *magic number* che identifica gli *script*, cioè quei file comandi non in linguaggio macchina che possono essere compresi solo con l'ausilio di un *interprete*.

/bin/bash è appunto l'interprete da invocare per gli script shell, ma il meccanismo vale anche per altri linguaggi (ad es. #!/usr/bin/perl)

Struttura degli script

I caratteri contenuti in uno script vengono interpretati dalla shell come se fossero digitati dall'utente. Ovviamente la possibilità di codificare in uno script comandi anche complessi richiede la disponibilità di strutture di controllo per eseguire test o per iterare azioni ripetitive. Tutte le strutture accettate da uno script possono comunque essere utilizzate sulla linea di comando.

Ogni script può inoltre accedere agli argomenti indicati sulla propria linea di comando, utilizzando le variabili:

$0, $1, $2, ...	comando, primo arg, secondo arg, …
"$*" / "$@"	"$1 $2 …" / "$1" "$2" …
$#	numero di argomenti

Verifica di condizioni

La verifica di una condizione e la specifica di una o più azioni da eseguire in modo condizionale si ottiene con il costrutto:

```
if <comando test 1>
then
    <comandi se test 1 ok>
[ elif <comando test 2>
then
    <comandi se test 2 ok> ]
[ else
        <comandi se nessun test ok> ]
fi
```

se il "comando test" restituisce un valore di ritorno =0 (OK), allora vengono eseguiti i comandi indicati nella clausola **then**.

Il case

La verifica di condizioni multiple è realizzabile più leggibilmente che con una catena di **elif** con il costrutto **case**:

```
case "$variabile" in
    nome1       ) echo vale nome1 ;;
    nome?       ) echo vale nome2, nomea, nomez ;;
    nome*       ) echo vale nome11, nome, nomepippo ;;
    [1-9]nome   ) echo vale 1nome, 2nome, …, 9nome ;;
    *           ) echo non cade in nessuna delle precedenti ;;
esac
```


Test

Il comando che viene eseguito per verificare una condizione if è spesso **test**. L'invocazione del test può essere abbreviata con il simbolo **[**

```
if test -e $filename
then
    rm $filename
fi
```

```
if [ -e $filename ]
then
    rm $filename
fi
```


Test unari

-d file	true se file esiste ed è una directory
-e file	true se file esiste
-f file	true se file esiste ed è un file regolare
-r file	true se file esiste ed è leggibile
-s file	true se file esiste ed ha dimensione >0
-w file	true se file esiste ed è scrivibile
-x file	true se file esiste ed è eseguibile
-z string	true se la stringa è vuota
-n string	true se string è non vuota

Test binari

file1 -nt file2 true se file1 è più 'nuovo' di file2
file1 -ot file2 true se file1 è più vecchio di file2
string1=string2 true se le stringhe sono uguali
string1 != string2 true se sono diverse
arg1 OP arg2 verifiche aritmetiche sugli operandi arg1 e arg2 - OP può valere:

- **-eq** uguaglianza numerica
- **-ne** diversità numerica
- **-lt** arg1 minore (strettamente) di arg2
- **-le** arg1 minore o uguale ad arg2
- **-gt** arg1 maggiore (strettamente) di arg2
- **-ge** arg2 maggiore o uguale ad arg2

Cicli definiti

E' possibile eseguire ripetutamente un insieme di comandi assegnando, volta per volta, ad una variabile un valore prelevato da una lista:

```
for numero in 1 2 3 4 5
do
    echo $numero
done
```

La lista di valori può essere anche omessa, in tal caso la variabile assume a turno il valore degli argomenti passati all'invocazione dello script.

Cicli definiti

Ricordiamo che la shell espande ogni stringa con wildcard nell'elenco dei nomi di file che concordano con tale schema. Con il **for** questo è molto utile:

```
for filename in *        # ←  esegue il ciclo per ogni file della
do                       #    directory corrente
    echo $filename       # ←  cosa succede se un file ha un
done                     #    carattere speciale nel nome?
```


Cicli indefiniti

Il ciclo for è un ciclo enumerativo, nel senso che il numero di iterazioni ed i relativi valori sono assegnati come lista. La shell consente anche l'uso di cicli non enumerativi che, al contrario del for, eseguono l'iterazione fino all'avverarsi di una condizione.
I cicli non enumerativi sono **while** ed **until**.

```
while [ $i -le 10 ]        oppure        until [ $i -gt 10 ]
do
    echo $i
    i=`expr $i + 1`
done
```

L'unica differenza è la negazione della condizione di uscita.

Interruzione forzata dei cicli

E' possibile uscire anticipatamente da un ciclo utilizzando la keyword **break**.

```
while true
do
    if [ $i -gt 10 ]
    then
        break
    fi
    echo $i
done
```

All'interno di un ciclo si puo' trovare anche **continue** per forzare l'esecuzione immediata dell'iterazione successiva.

Le funzioni

Il linguaggio di script della shell consente anche le definizione di funzioni per la modularizzazione delle attività. Una funzione può anche utilizzare variabili locali:

```
f1 () {
    local a=10
    echo $a
}

f1              # scrive '10'
echo $a         # non scrive nulla
```


Command substitution.

È possibile valutare 'al volo' l'output di un comando racchiudendolo tra *backtick* (virgolette a rovescio `).

```
for filename in `ls p*`
do
    echo $filename
done
```

```
dottori=`cat /tmp/elenco | grep Dr. | sort -u`
```


Utilità di impiego comune

Filtri di manipolazione del testo

Ricerca di file

L'editor "vi"

Ricerca di parti in un testo: Grep

grep è forse la più nota delle utility.

Esamina le righe del testo in ingresso (su standard input o specificato come elenco di file sulla riga di comando), e riproduce in uscita quelle che contengono l'*espressione regolare* (nel caso più semplice una sottostringa) passata come argomento. Es. per cercare un file di nome "prova" all'interno dell'output di ls:

ls | grep prova

Espressioni regolari

L'utility grep (ma non solo) utilizza una sintassi per la specifica dei *pattern* di ricerca detta espressione regolare (regexp). All'interno di una regexp:

. indica qualsiasi carattere
[abc] indica qualsiasi carattere fra *a*, *b* o *c*
[a-z] indica qualsiasi carattere fra *a* e *z* compresi
[^dc] indica qualsiasi carattere che non sia né *d* né *c*
? indica zero o una occorrenza di cio' che lo precede
***** indica zero o più occorrenze di ciò che lo precede
+ indica una o più occorrenze di ciò che lo precede
^ indica l'inizio della linea
$ indica la fine della linea

Espressioni regolari - esempi

grep '^att' miofile
ha come output tutte le righe di miofile che iniziano per **att**

grep '^Nel.*vita\.$' miofile
ha come output tutte le righe di miofile che iniziano per **Nel** e finiscono per **vita.**

grep '.es.e' miofile
righe che contengono 1 carattere seguito da **es** seguito da 1 carattere seguito da **e**

Grep – opzioni principali

Si possono alterare la modalità di ricerca di grep e l'aspetto dell'output per mezzo di opzioni da specificare sulla riga di comando prima della regexp e degli eventuali file.

-i rende l'espressione insensibile a maiuscole e minuscole
-v grep restituisce le linee che **non** contengono l'espressione

-l utile se passo a grep più file su cui cercare: restituisce solo i nomi dei file in cui l'espressione è stata trovata
-n restituisce anche il numero della riga contenente l'espressione

Estrazione dei campi – cut

Il comando **cut** permette di *tagliare* parti di righe. Su file organizzati a 'record' (linee) per i quali ogni record rappresenta una lista di 'campi' opportunamente delimitati, permette quindi di estrarre uno o più campi di ciascun record.

cut –c*elenco_caratteri* ritaglia le righe contando i caratteri

Es: **cut –c15 file** restituisce solo il 15° carattere
cut –c8–30 file restituisce i caratteri dall'8° al 30°
cut –c–30 file restituisce i caratteri fino al trentesimo
cut –c8– file restituisce i caratteri dall'ottavo in poi

Estrazione dei campi – cut

cut –d*carattere_delimitatore* **–f***elenco_campi*
ritaglia le righe dove trova il delimitatore, generando una lista di campi

Es. se ci interessa estrarre solo il campo username dal file passwd:

cat /etc/passwd | cut –f1 –d: –s

Es. se nel campo note metto 'Nome Cognome' e voglio l'iniziale dei cognomi degli utenti:

cat /etc/passwd | cut –f5 –d: –s | cut –f2 –d' ' | cut –c1

Il comando rev

rev è un filtro che permette di invertire l'ordine dei caratteri di ogni linea dello stream in input verso lo stream di output.

L'utilità del comando è solitamente quella di accompagnare cut nella estrazione di campi la cui posizione sia nota relativamente al fine linea:

cat /etc/passwd | rev | cut -f1 -d: -s | rev

elabora il contenuto del file /etc/passwd nel sequente modo:
- inverte ogni linea
- prende il primo campo (→ l'ultimo campo dell'originale)
- inverte ogni linea (ripristina il campo selezionato)

I comandi sort e uniq

Per ordinare le linee di uno stream o per individuare le righe duplicate o uniche, unix mette a disposizione i filtri sort e uniq. Al solito, questi filtri operano sullo standard input se non c'è nessun argomento.

sort ordina alfabeticamente
sort –n ordina numericamente
sort –u elimina le entry multiple (equivale a sort | uniq)
sort –r inverte l'ordine

uniq elimina i duplicati
uniq –c indica anche il numero di occorrenze
uniq –d mostra solo le entry non singole

I comandi head e tail

head -N è un filtro che permette di estrarre solo le prime *N* righe di un file

tail -N è un filtro che permette di estrarre solo le ultime *N* righe di un file

Un'opzione particolare di **tail** è **–f** con cui, dopo aver mostrato le ultime righe di un file, lo si mantiene aperto e si visualizzano in tempo reale eventuali nuove righe che vi vengano appese da altri processi

Il comando wc

wc (word count) è un filtro di conteggio

wc -c conta i caratteri
wc -l conta le linee
wc -w conta le parole (stringhe separate da spazi)

Modifiche più complesse

Esistono altri comandi che consentono operazioni complesse sui file, come sed e awk. È riduttivo chiamarli filtri, dispongono di un vero e proprio linguaggio di programmazione per effettuare operazioni di manipolazione del testo. Vediamo solo qualche esempio pratico di utilità frequente:

sed **s/***vecchio_pattern***/***nuovo_valore***/** sostituisce in ogni riga un nuovo valore ad una parte di testo coincidente con un pattern

Es. inserisce la stringa "Linea:" all'inizio di ogni riga di passwd:
cat /etc/passwd | sed 's/^/Linea:/'
Abbrevia il titolo di dottore, accetta minuscolo o maiuscolo:
cat personale | sed 's/dottore/Dott./i'

Modifiche più complesse

awk è usato come evoluzione di cut, perché permette di considerare qualsiasi sequenza di spazi e tab (blanks) come un unico delimitatore, mentre, ad esempio, **cut –f2 –d' '** se ci sono 2 spazi dopo il primo campo considera il 2° spazio come 2° campo.

Es. stampa il secondo campo del file, purchè sia separato dal primo da un numero qualunque di blanks
cat personale | awk '{print $2}'

Ricerca di file

Il filesystem di unix/Linux, specialmente in installazioni di tipo server, ospita comunemente centinaia di migliaia di file. Cercare un file o una directory all'interno di un insieme così vasto di file può essere una operazione frustrante, nonostante l'agevolazione costituita dall'ordine gerarchico del filesystem e dalle linee guida standard per la collocazione delle risorse (Es. se stiamo cercando un file di configurazione, con ogni probabilità questo si trova sotto /etc)

Sono per questo a disposizione due utility di ricerca molto potenti, una ottimizzata per la flessibilità, l'altra per la velocità.

Ricerca di file - find

find ricerca in tempo reale i file che soddisfano una combinazione di criteri, ad esempio file che:

- abbiano un nome che contenga una espressione data
- siano stati usati in un periodo specificato
- abbiano dimensione compresa tra un minimo e un massimo
- siano di un tipo specificato (file, dir, link simbolici, ...)
- siano di proprietà di un utente o di un gruppo specificati
- abbiano certi permessi di accesso
- si trovino ad una certa profondità dell'albero del filesystem
- soddisfino una o più delle precedenti condizioni

Uso di find

il programma find si utilizza con la seguente convenzione:

find [FILE...] [EXPRESSION]

dove FILE rappresenta il o i percorsi di ricerca e EXPRESSION indica l'elenco delle caratteristiche richieste o delle operazioni desiderate.

Es. per ricercare sotto /usr/src tutti i file che finiscono per '.c', hanno dimensione maggiore di 100K, ed elencarli sullo standard output:

find /usr/src -name '*.c' -size +100k -print

Uso di find

Una delle opzioni più potenti di find permette, per ciascun oggetto individuato secondo i criteri impostati, di invocare l'esecuzione di un comando:

Es. mostra il contenuto dei file trovati
find /usr/src -name '*.c' -size +100k **-exec** cat **{} \;**

il comando che segue **–exec** viene lanciato per ogni file trovato, la sequenza **{}** viene sostituita di volta in volta con il nome del file, **\;** è necessario per indicare a find la fine del comando da eseguire.

Es. elenca solo i file che contengono *stringadacercare*
find / -type f **-exec** grep -l stringadacercare **{} \;**

Ricerca di file - locate

find è molto potente e flessibile, ma lanciato su filesystem grandi è molto lento poiché percorre le directory una per una.

locate ricerca file che hanno un certo pattern nel nome senza accedere direttamente a tutte le directory del filesystem coinvolte ma utilizzando un database di file che opera da 'cache'. È quindi molto più veloce di find, ma "fotografa" la situazione all'istante in cui il database è viene aggiornato (mediante l'utility **updatedb**) e non permette di filtrare la ricerca con criteri flessibili come quelli di find.

Cstruzione di linee di comando complesse con xargs

xargs <comando> si aspetta sullo standard input un elenco di stringhe, ed invoca poi **comando** con tali stringhe come argomenti. L'uso di xargs permette il passaggio di un numero qualsiasi di argomenti (se si eccede la lunghezza massima della command line, xargs divide in più invocazioni e raggruppa le invocazioni in modo da ridurre il carico del sistema.)

Es. alternativamente a **–exec** dell'esempio di find:
find /usr/src -name '*.c' -size +100k -print | xargs cat

Cstruzione di linee di comando complesse con xargs

Es. manda un messaggio di posta vuoto con oggetto "guasto! uscite" a tutti gli utenti:
cat /etc/passwd | cut –f1 –d: | xargs mail –s 'guasto! uscite'

Nota: si poteva anche con i backtick:
mail –s 'guasto! uscite' `cat /etc/passwd | cut –f1 –d:`

xargs ha però molte opzioni che lo rendono più flessibile:
- sostituzione di parti degli argomenti e definizione dei delimitatori di argomento
- esecuzione interattiva con conferma di ogni linea di comando generata
- definizione dei limiti di numero di argomenti e di lunghezza delle command line

Vi

Il funzionamento di una macchina unix è fortemente influenzato da diversi file di configurazione in formato 'testo'. La creazione e la modifica di tali file puo' essere fatta da linea di comando (ridirezione, line editor, ...) o, più comodamente, utilizzando un text editor. Sono disponibili per unix/Linux diversi editor, sia utilizzabili da console che da ambiente grafico, che possono essere installati dalle varie distribuzioni. Tipicamente viene installato l'editor vi che si contraddistingue per la sua semplicità e per la ridotta occupazione di risorse.La versione 'storica' di vi e' spesso sostituita da enhancement come vim o elvis che aggiungono funzionalita' di visualizzazione o editing. Non si deve essere tratti in inganno dall'interfaccia utente spartana, le potenzialità di gestione del testo di vi sono strabilianti.

VI

Il vi e' un editor a 'tutto schermo' che utilizza una interfaccia 'modale'. In altre parole, il programma si puo trovare in uno dei seguenti stati:

COMMAND	Il cursore è posizionato sul testo, la tastiera è utilizzabile solo per richiedere l'esecuzione di comandi, e non per introdurre testo. I caratteri digitati non vengono visualizzati.
INPUT	Tutti i caratteri digitati vengono visualizzati ed inseriti nel testo.
DIRECTIVE	Ci si trova posizionati con il cursore nella linea direttive (l'ultima linea del video) e si possono richiedere tutti i comandi per il controllo del file.

Passaggi di stato

I passaggi di stato avvengono digitando i seguenti caratteri:

DIRECTIVE MODE	----->	COMMAND MODE	<RET>
COMMAND MODE	----->	INPUT MODE	oORiaA
INPUT MODE	----->	COMMAND MODE	<ESC>
COMMAND MODE	----->	DIRECTIVE MODE	:/?

Comandi di spostamento

In COMMAND MODE, e' possibile richiedere lo spostamento del 'cursore' con i comandi di movimento:

- h 1 spazio a sinistra (come backspace)
- l 1 spazio a destra (come space)
- k 1 linea sopra (stessa colonna)
- j 1 linea sotto (stessa colonna)

(in sostituzione di questi quattro caratteri, si possono anche utilizzare le frecce per spostarsi nelle diverse direzioni)

- G posizionamento sull'ultima linea del testo
- #G posizionamento sulla linea identificata da "#"

Esempio: 3G 5G 175G

Modifica

I comandi di spostamento consentono di visualizzare il contenuto di un file; in command mode è possibile apportare delle modifiche al file:

cancellare:
- x cancella il carattere su cui si trova il cursore
- dd cancella la linea su cui si trova il cursore

inserire (provoca l'ingresso in INPUT MODE):
- i entra in modalità 'inserimento' nella posizione del cursore
- a entra in modalità 'append' nella posizione del cursore
- A entra in modalità 'append' a fine riga
- R entra in modalità 'replace'
- o inserisce una linea vuota sotto al cursore
- O inserisce una linea vuota sopra al cursore

Comandi di gestione file

<ESC> esce dalla modalita' INPUT

I comandi "directive" per salvare/uscire sono:
:w scrive il file corrente
:q esce (:q! se si vuole uscire senza salvare)
ZZ scrive ed esce

Ricerca e sostituzione

/string cerca la prossima linea che contiene string
?string cerca la linea precedente che contiene string
/ e ? senza string ripetono la ricerca precedente

string può essere anche una espressione regolare.

:s/trova/sostituisci/ cerca 'trova' nella linea corrente e lo sostituisce con 'sostituisci'

:%s/trova/sostituisci/cgi cerca 'trova' in ogni linea del file e lo sostituisce con 'sostituisci' dopo aver chiesto conferma (c), anche più volte nella stessa linea (g), case-insensitive (i)

Copia & incolla

i comandi di copia utilizzano dei buffer interni a vi; tipicamente si utilizza il buffer standard, ma e' possibile specificarne altri:

yy copia la linea corrente nel buffer
"ayy copia la linea corrente nel buffer "a"

per copiare gruppi di linee si può procedere nei seguenti modi:

- ci si posiziona sulla prima linea del gruppo
- **ma** marca la posizione con il simbolo "a"
- ci si posizione sull'ultima linea del gruppo
- **y'a** copio nel buffer dalla posizione corrente a quella marcata "a"

Copia & incolla

…oppure:
- ci si posiziona sulla prima linea del gruppo
- per copiare 10 righe si batte **10yy**

…oppure (molto potente):
- ci si posiziona sulla prima linea del gruppo
- per copiare fino alla parola 'basta' (esclusa) si batte
y/basta<RET>

Il paste (incolla) si ottiene con i comandi:
p incolla dopo la linea corrente
P incolla prima della linea corrente

Configurazione e monitoraggio

Avvio e arresto dei servizi e del sistema

Esecuzione programmata di comandi

Monitoraggio off-line: i "diari di bordo"

Comandi di monitoraggio real-time

Avvio e arresto dei servizi

Un concetto importante per i sistemi Linux è quello di *runlevel*. Possiamo definire il runlevel come una situazione di funzionamento caratterizzata dall'attività di certi servizi e dalla quiescenza di altri.
I runlevel standard sono:

0	Halt
1	Single user mode
2	Multiuser mode, senza NFS
3	Multiuser mode, completo
4	Non usato
5	Multiuser mode completo con login sotto X
6	Reboot

Avvio e arresto dei servizi

Tutti i file di configurazione che permettono di controllare i runlevel risiedono nella directory **/etc/rc.d**. In particolare vi si trovano:

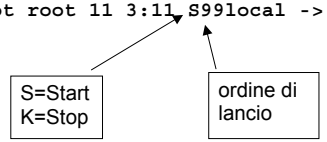
- il file *rc.sysinit* [,*rc.local, rc.serial*]
- le directory *init.d, rc0.d, rc1.d, rc2.d, rc3.d, rc4.d, rc5.d, rc6.d* [,*rc.local, rc.serial*]

init.d contiene uno script per il lancio e l'arresto di ciascuno dei servizi installati, esclusi quelli che devono essere eseguiti una volta sola indipendentemente dal runlevel. Questi script devono essere messi in *rc.local* o *rc.serial*.

Avvio e arresto dei servizi

Le directory rc*N*.d servono da raccoglitori delle istruzioni di avvio o arresto dei servizi specifiche di ciascun runlevel. Se ad esempio si esamina il contenuto della directory *rc3.d*, si troverà qualcosa di simile a questo:

```
lrwxrwxrwx 1 root root 17 3:11 S10network -> ../init.d/network
lrwxrwxrwx 1 root root 16 3:11 S30syslog -> ../init.d/syslog
lrwxrwxrwx 1 root root 14 3:32 S40cron -> ../init.d/cron
lrwxrwxrwx 1 root root 14 3:11 S50inet -> ../init.d/inet
lrwxrwxrwx 1 root root 13 3:11 S60nfs -> ../init.d/nfs
lrwxrwxrwx 1 root root 15 3:11 S70nfsfs -> ../init.d/nfsfs
lrwxrwxrwx 1 root root 18 3:11 S90lpd -> ../init.d/lpd.init
lrwxrwxrwx 1 root root 11 3:11 S99local -> ../rc.local
```



Avvio e arresto dei servizi

Nelle directory rc*N*.d non ci sono file "reali", ma solo link agli script in *init.d*, che sono fatti in modo da accettare un parametro che può essere **start** o **stop**. Di fatto gli script in init.d sono il modo migliore per controllare anche a mano i processi.

Es:*/etc/rc.d/init.d/httpd start*

Per RedHat Linux l'utility ***chkconfig*** fornisce una semplice intrfaccia a linea di comando per mantenere la gerarchia */etc/rc.d* senza doversi preoccupare di tutti i link.

Esercizio: realizzare uno script shell che elenchi in quali runlevel è attivato un servizio passato come argomento

La procedura di avvio e arresto del sistema

L'avvio e l'arresto dell'intero sistema è coordinato dal processo ***init*** che decide il runlevel di default e utilizza gli script definiti in precedenza per gestire i servizi.

init è il primo processo eseguito dal kernel al boot (lo si trova sempre con PID=1). Il suo comportamento è configurato dal file ***/etc/inittab***.

/etc/inittab

Il file /etc/inittab e' strutturato come elenco di record delimitati dal fine linea (ogni linea e' un record). Ogni record rappresenta una entry divisa in campi dal delimitatore ':'

id:runlevels:action:process

id	identificativo unico della entry
runlevels	lista dei runlevel in cui la entry va processata
action	specializzazione dell'azione da compiere
process	processo che deve essere eseguito (eseguibile)

Esecuzione programmata

Nell'amministrazione di un sistema l'esecuzione di programmi di monitoraggio è un'esigenza primaria.
Tre modalità di azione possono essere distinte:

- eseguire un programma con continuità
- eseguire un programma con cadenza periodica
- eseguire un programma non in modo ripetitivo ma in un momento ben preciso

Una entry del tipo *xx*:345:**respawn**:/usr/bin/myprocess in *inittab* garantisce che *init* avvii myprocess e lo sorvegli. Per le altre due esigenze si impiegano rispettivamente il ***cron daemon*** e l'***at daemon***.

Cron

crond è un demone (programma residente in memoria in attesa di eventi che lo riguardino) che esamina una serie di file di configurazione ogni minuto, e determina quali compiti specificati nei file debbano essere eseguiti.

I file di configurazione (crontab) sono distinti in due insiemi:

- Uno per utente (***/var/spool/cron/<utente>***) → *crontab –e*
- System-wide (***/etc/crontab***)

Solitamente quest'ultimo non fa altro che richiamare l'esecuzione di tutto ciò che trova in alcune directory: */etc/cron.hourly/*, */etc/cron.daily/*, */etc/cron.weekly/*, */etc/cron.monthly/*

Cron

Ogni crontab contiene un elenco di direttive nella forma

```
MINUTO ORA G.MESE MESE G.SETTIMANA   <comando>
```

Es.

```
*    *       27      *     *     $HOME/bin/paga
30   8-18/2  *       *     1-5   $HOME/bin/lavora
00   00      1       1     *     /usr/sbin/auguri
30   4       1,15    *     6     /bin/backup
```


At

atd è un demone che gestisce code di compiti da svolgere in momenti prefissati. L'interfaccia ad *atd* consiste di 4 comandi:

at [-V] [-q queue] [-f file] [-mldbv] TIME
pianifica un comando al tempo TIME
atq [-V] [-q queue] [-v]
elenca i comandi in coda
atrm [-V] job [job...]
rimuove comandi dalla coda
batch [-V] [-q queue] [-f file] [-mv] [TIME]
esecuzione condizionata al carico

At

- Se non viene specificato un file comandi per *at* o *batch*, verrà usato lo standard input.

- La specifica dell'ora è flessibile e complessa. Per una definizione completa si veda la documentazione in ***/usr/doc/at-<versione>/timespec***. Alcuni esempi:

```
echo 'wall "sveglia"' | at 08:00
echo "$HOME/bin/pulisci" | at now + 2 weeks
echo "$HOME/bin/auguri" | at midnight 25.12.2003
```

Diagnostica di sistema

La maggior parte dei servizi gira in *background*, ovvero senza essere collegata ad un terminale da cui ricevere input o su cui scrivere output. Per evitare di doversi "inventare" per ogni servizio un gestore dell'output diagnostico e di registrazione delle azioni importanti, e per avere un metodo uniforme di classificazione dei messaggi prodotti dal sistema e dagli applicativi, si utilizza ***syslogd***.

Syslogd utilizza le direttive di configurazione in /etc/syslog.conf per decidere la destinazione di ogni messaggio che gli viene inviato, sulla base di un'etichetta che specifica una facility e una priorità

/etc/syslog.conf

```
# destinazione: file
kern.*                  /dev/console
*.info;mail.none;       /var/log/messages

# destinazione: utenti collegati
*.emerg                 *

# destinazione: syslogd di un altro sistema
*.emerg                 @loghost
```


Diagnostica di sistema

Qualunque programma può sfruttare *syslogd*, per mezzo di una chiamata di sistema in C alla funzione ***syslog*** o per mezzo dell'utility ***logger***.

logger –p <priorità> messaggio

È possibile definire facility personalizzate nel file */etc/syslog.conf*, chiamate ***local0…loca7***

Nota importante: qualunque metodo usi un'applicazione per produrre dati diagnostici e di controllo, bisogna prevedere un metodo **automatico** per tenere sotto controllo la dimensione dei file che ne derivano.

Monitoraggio delle risorse

Un ampio set di comandi consente di ispezionare lo stato delle risorse di un sistema linux in tempo reale e di visualizzare statistiche elaborate su dati raccolti con continuità:

Processi	Utenti	Spazio	File
ps top uptime	w last	df du free vmstat	fuser lsof

Monitoraggio delle risorse – ps

Il già citato *ps* consente di "fotografare" l'elenco dei processi attivi al momento dell'invocazione, con i dettagli di ognuno.

Esercizio: Registrare in un logfile il numero di demoni *httpd* attivi, con una frequenza di campionamento di 1 minuto. Variante: abbassare la frequenza di campionamento a 5 s.

Esercizio: avvertire immediatamente *root* appena l'utente xxx lancia un qualunque processo

Monitoraggio delle risorse – uptime e free

uptime restituisce informazioni molto più sintetiche sulla "storia di carico" del sistema, così come ***free*** riporta statistiche sulla memoria. La loro invocazione dà il "polso" del sistema:

```
# uptime
9:14am  up 33 min,  2 users,  load average: 0.02, 0.03, 0.13

# free -o
             total       used       free     shared    buffers     cached
Mem:        384480     379276       5204       1312      51284     138752
Swap:       128516          0     128516
```

La maggior parte di queste informazioni è accessibile tramite il filesystem speciale ***/proc***, questi comandi sono semplicemente interfacce utente.

Monitoraggio delle risorse – top

top è una interfaccia allo stesso tipo di informazioni più adatta alla sorveglianza in tempo reale, in quanto:

- Aggiorna l'elenco con cadenza configurabile
- Raccoglie lo stato complessivo del sistema e lo stato dettagliato dei porcessi in un'unica presentazione
- Ordina l'elenco dei processi con criteri configurabili
- Permette la variazione dei criteri di visualizzazione con comandi interattivi, runtime
- Permette l'invio di segnali ai processi

… sostanzialmente raggruppando le funzionalità di ps, uptime, free e kill in un unico strumento interattivo

Monitoraggio delle risorse – top

```
  9:31am  up 50 min,  2 users,  load average: 0.02, 0.02, 0.04
71 processes: 70 sleeping, 1 running, 0 zombie, 0 stopped
CPU states:  4.3% user,  5.2% system,  0.1% nice, 90.2% idle
Mem:   384480K av,  380688K used,    3792K free,    1312K shrd,   51312K buff
Swap:  128516K av,       0K used,  128516K free                  139136K cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
 1179 root      13   0  3092 3092  2592 S     2.8  0.8   0:50 magicdev
 9299 root      16   0  1044 1040   832 R     2.8  0.2   0:00 top
    1 root       8   0   520  520   452 S     0.0  0.1   0:03 init
    2 root       9   0     0    0     0 SW    0.0  0.0   0:00 keventd
    3 root       9   0     0    0     0 SW    0.0  0.0   0:00 kapm-idled
    4 root      19  19     0    0     0 SWN   0.0  0.0   0:00 ksoftirqd_CPU0
    5 root       9   0     0    0     0 SW    0.0  0.0   0:00 kswapd
    6 root       9   0     0    0     0 SW    0.0  0.0   0:00 kreclaimd
    7 root       9   0     0    0     0 SW    0.0  0.0   0:00 bdflush
    8 root       9   0     0    0     0 SW    0.0  0.0   0:00 kupdated
    9 root      -1 -20     0    0     0 SW<   0.0  0.0   0:00 mdrecoveryd
   71 root       9   0     0    0     0 SW    0.0  0.0   0:00 khubd
  465 root       9   0     0    0     0 SW    0.0  0.0   0:00 eth0
  546 root       9   0   592  592   496 S     0.0  0.1   0:00 syslogd
  551 root       9   0  1124 1124   448 S     0.0  0.2   0:00 klogd
  569 rpc        9   0   592  592   504 S     0.0  0.1   0:00 portmap
  597 rpcuser    9   0   788  788   688 S     0.0  0.2   0:00 rpc.statd
```

Monitoraggio delle risorse – df

df mostra l'utilizzo dello spazio disco:

```
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/hdb1               202220    126789     64991  67% /
/dev/hdb4              5558076   3916612   1359124  75% /usr
/dev/hdb3               303336     49822    237851  18% /var
none                    192240         0    192240   0% /dev/shm
/dev/hda1             10231392   9473248    758144  93% /win/c
/dev/hda5              9790032   4247000   5543032  44% /win/d
```

Esercizio: avvertire *root* quando lo spazio libero di qualunque partizione scende sotto il 10%
Variante: lo script accetta un parametro che indica quale partizione sorvegliare, se il parametro è assente → tutte

Monitoraggio delle risorse – du

du permette di calcolare lo spazio occupato dai file (in una directory). Senza opzioni particolari **du** riporta l'occupazione totale delle dir passate come argomento ed anche di tutte le subdir in esse presenti. Es:

```
# du /tmp
1   /tmp/.font-unix
1   /tmp/.X11-unix
1   /tmp/.ICE-unix
5   /tmp/orbit-root
72  /tmp
```

du –s riporta invece il *summary*, senza dettagli sulle subdir.

Esercizio: a mezzanotte inviare a root un rapporto sulle dir più ingombranti: 10 di primo livello, 20 di secondo, 30 di terzo.

Monitoraggio delle risorse – fuser

I processi ed i file sono legati da due relazioni d'uso interessanti per l'amministratore: quali file sta usando un processo e quali processi stanno usando un file.

Alla prima domanda si può rispondere esaminando il filesystem speciale **/proc**, es.:

```
# ls -l /proc/2208/fd/
total 0
lrwx------    1 root     root     64 Apr 26 10:11 0 -> /dev/pts/0
lrwx------    1 root     root     64 Apr 26 10:11 1 -> /dev/pts/0
lrwx------    1 root     root     64 Apr 26 10:11 2 -> /dev/pts/0
lr-x------    1 root     root     64 Apr 26 10:11 3 -> /etc/man.config
```

Esercizio: elencare tutti i file impegnati da un utente per mezzo dei suoi processi

Monitoraggio delle risorse – fuser

Il comando ***fuser*** calcola la risposta alla seconda domanda, e la visualizza insieme ad utili informazioni accessorie:

```
# fuser /etc/man.config
/etc/man.config:      2208    2212    2213    2219
```

fuser permette anche di indagare un intero filesystem:

```
# fuser -m /var
/var:                  546    597c    714    714c    879c    898    916
916c    964   1013   1020   1021   1318   6493   9244   9244m   9249   9249m
9275   9275m

c       current directory.
e       executable being run.
f       open  file.  f is omitted in default display mode.
r       root directory.
m       mmap'ed file or shared library.
```