Introduzione alla shell UNIX

Laboratorio Sistemi Operativi

Giuseppe Salvi
Email: giuseppe.salvi@uniparthenope.it

Definizione di shell

- La shell è un programma speciale usato come interfaccia tra l'utente ed il kernel del sistema operativo UNIX/Linux
- Il kernel è caricato in memoria in fase di boot e gestisce il sistema finché resta in opera
  - Crea e controlla i processi, gestisce la memoria, i file, la comunicazione …
  - Tutti gli altri programmi, inclusa la shell, risiedono su disco
    - Il kernel carica tali programmi in memoria, li esegue, e ripulisce il sistema una volta che essi terminano
- La shell è un programma di utilità che è avviato una volta effettuato il log nel sistema
  - Consente l'interazione con il kernel interpretando i comandi digitati da riga di comando o contenuti in uno script

Funzionalità di una shell

- Dopo il login, è avviata la shell che è in attesa di ricevere input dall'utente
  - Immesso un comando, è responsabilità della shell
    - Fare il parsing della riga di comando
    - Gestire i simboli wildcard, la redirezione, le pipe ed il controllo dei job
    - Cercare un comando e, se trovato, eseguire il comando
  - Tipicamente, imparando UNIX/Linux, si trascorre molto tempo ad eseguire comandi dal prompt
    - Si usa la shell interattivamente
  - Se si esegue spesso lo stesso insieme di comandi in modo ripetitivo e regolare risulta utile automatizzare una tale attività
    - Ciò è possibile inserendo i comandi in un file, chiamato script, ed eseguendo, successivamente, il file di script
    - Gli script possono essere anche piuttosto complessi e contenere, oltre ai comandi da eseguire, costrutti di programmazione per prendere decisioni, cicli, test su file, ecc.

Le shell di UNIX

- Esistono tre shell predominanti sui principali sistemi UNIX
  - La shell Bourne (AT&T)
  - La shell C (Berkeley)
  - La shell Korn (un’estensione della shell Bourne)
- Tutte si comportano più o meno allo stesso modo quando usate in modo interattivo
  - Si differenziano maggiormente nella sintassi e nell’efficienza quando usate come linguaggio di scripting
- La shell Bourne (sh) è la shell standard di UNIX, usata per amministrare il sistema
  - Scritta presso il lab della AT&T è nota per essere concisa, compatta e veloce
  - Il prompt di default della shell Bourne è il $

Le shell di UNIX(2)

- La shell C (csh), sviluppata a Berkeley, aggiunge numerose caratteristiche quali la storia della riga di comando, l'aliasing, aritmetica built-in, il completamento dei nomi di file ed il controllo dei job
- Ha avuto maggiori consensi rispetto alla Bourne se usata in modo interattivo
  - Al contrario, per lo scripting è preferita la Bourne poiché gli script sono più semplici e più veloci
- Il prompt di default e il %

Le shell di UNIX (3)

- La shell Korn (ksh), scritta nei lab AT&T da David Korn, include le funzionalità della shell Bourne ed introduce un numero di caratteristiche aggiuntive
  - Storia editabile, alias, funzioni, wildcard per espressioni regolari, aritmetica built-in, controllo dei job, ecc.
- La shell Korn è quasi completamente compatibile con la shell Bourne
- Il prompt di default è il $
- Esiste anche una versione di pubblico dominio (pdksh) disponibile per diverse piattaforme (Linux incluso)

Le shell di Linux

- La shell GNU Bourne Again o Bash, è la shell di default di Linux e risulta essere un'estensione della shell Bourne
  - Sia a livello di programmazione che di uso interattivo
    - All'utente è consentito di adattare il proprio ambiente di lavoro
- E' una delle più popolari shell usate dagli utenti UNIX e Linux ed è conforme allo standard POSIX
- Il prompt di default è il $
- Un'altra shell ampiamente usata dagli utenti Linux è la shell TC (tcshell) una ramificazione compatibile della shell C di UNIX, con numerose caratteristiche aggiuntive
- Il suo prompt di default è il >
- Un'ulteriore shell di Linux è la shell Z (zsh) che incorpora caratteristiche delle shell Bourne Again, TC e Korn
- Per verificare quali shell sono disponibili nella versione Linux che si usa esaminare il contenuto del file /etc/shells
  - cat /etc/shells

Usi della shell: interprete dei comandi

- Quando usata in modo interattivo una delle principali funzioni della shell è di interpretare i comandi immessi al prompt
  - La shell analizza sintatticamente la riga di comando, la divide in parole (chiamate token) separate da spazi
  - Se le parole contengono metacaratteri speciali, la shell li valuta
  - Dopo che la riga di comando è stata elaborata, la shell cerca il comando e ne avvia l'esecuzione
- La shell gestisce il file I/O e l'elaborazione in background

Usi della shell: personalizzazione ambiente utente

- Ciò viene fatto, normalmente, nei file di inizializzazione della shell
  - Questi file contengono le definizioni per impostare
    - il terminale e le caratteristiche della finestra
    - Le variabili che definiscono i percorsi di ricerca, permessi, prompt e tipo di terminale
    - Le variabili richieste per applicazioni specifiche (gestori di finestre, programmi per il text-processing e librerie per linguaggi di programmazione)
    - La Bash fornisce anche ulteriori personalizzazioni per l'aggiunta della storia, alias e variabili built-in impostate per proteggere l'utente dal danneggiare file, fare logout in modo inavvertito o notificare l'utente quando un job è completato

Usi della shell: linguaggio di programmazione interpretato

- I programmi di shell (script) consistono di comandi elencati in un file
- I programmi sono creati con un editor
  - Consistono di comandi UNIX/Linux inframmezzati con costrutti fondamentali di programmazione quali assegnamenti di variabili, test condizionali e cicli
  - Gli script non si compilano, la shell interpreta ogni linea dello script come se fosse stato immesso da tastiera

Startup del sistema e shell di Login

- Quando si avvia il sistema il primo processo invocato è init
  - Il pid di init è 1
  - init inizializza il sistema ed avvia gli altri processi per aprire le linee del terminale ed impostare gli standard input (stdin), output (stdout) ed error (stderr) che sono tutti associati al terminale
  - Lo standard input è associato alla tastiera, lo standard output e lo standard error vanno sul video
  - A questo punto appare il prompt dei comandi sul terminale
- Una volta inserito il nome utente viene richiesta la password
  - Il programma /bin/login verifica l'identità controllando il primo campo del file /etc/passwd
    - Se il nome corrisponde, la password immessa è fornita ad un programma di crittografia che verifica la sua correttezza
    - Verificata la password il programma login imposta l'ambiente iniziale che consiste di variabili che definiscono l'ambiente di lavoro passato alla shell

Startup del sistema e shell di Login (2)

- Alle variabili HOME, SHELL, USER e LOGNAME sono assegnati i valori estratti dal file /etc/passwd
- E’ impostata la variabile PATH alle directory in cui si trovano le utility usate più di frequente
- Quando il programma login ha terminato, avvia il programma trovato nell’ultima entrata del file passwd, la shell (shell di login)
- La shell controlla la directory home per verificare se esistono file di inizializzazione specifici della shell
  - Se esistono sono eseguiti. Tali file sono usati per personalizzare ulteriormente l’ambiente utente

Parsing della riga di comando

- Quando si inserisce un comando
  - La shell analizza la riga di comando e la divide in token
    - I token sono separati da spazi e la riga di comando è terminata dal newline
  - La shell controlla se la prima parola è un comando built-in o un programma eseguibile localizzato da qualche parte su disco
    - Se è un comando built-in la shell esegue il comando internamente
    - Altrimenti, ricerca all'interno delle directory specificate nella variabile PATH per sapere dove risiede il programma
      - Se il programma viene trovato, la shell fa il fork di un nuovo processo ed esegue il programma
      - La shell attenderà fino al termine dell'esecuzione e poi, se necessario, riporterà lo stato del programma terminante

Tipi di comandi

- Quando è eseguito un comando questo corrisponderà a un *alias*, una *funzione*, un comando *built-in*, o un programma *eseguibile* che risiede su disco
  - Gli alias sono abbreviazioni di nomi di comandi
  - Le funzioni sono gruppi di comandi organizzati in routine separate
    - Alias e funzioni sono definite nella memoria della shell
    - I comandi built-in sono routine interne alla shell
    - I programmi eseguibili risiedono su disco
  - La shell usa la variabile PATH per localizzare i programmi eseguibili su disco e crea un processo figlio prima che il comando sia eseguito
  - Quando la shell è pronta per eseguire il comando, essa valuta il tipo di comando secondo l'ordine
    - Alias
    - Keyword
    - Funzioni
    - Comandi built-in
    - Programmi eseguibili

I processi e la shell

- Un processo è un programma in esecuzione e può essere identificato dal suo PID univoco
- Il kernel controlla e gestisce i processi
- Un processo consiste di un *programma eseguibile*, i suoi *dati* ed il suo *stack*, i *registri* e tutte le informazioni di cui ha bisogno il programma per l'esecuzione
- La shell è un programma speciale che è avviato dopo il processo di login
- Una volta avviata, la shell è un processo

I processi in esecuzione

- Il comando ps
  - Con le sue molteplici opzioni (*man 1 ps* oppure *man ps*) visualizza un elenco di processi correntemente in esecuzione in diversi formati possibili
  - Es.: al prompt dei comandi digitare *ps aux*
  - Un altro modo per esaminare quali processi sono in esecuzione e quali processi sono processi figli è fornito dal comando pstree
    - Visualizza tutti i processi come un albero,
    - Se è specificato un nome utente, i processi di tale utente saranno alla radice dell'albero

Sintassi di ps

- Il comando `ps` fornisce le informazioni su ogni processo in esecuzione. La sintassi è la seguente: `ps [options]`
  - Alcune possibili opzioni sono:
    - a : mostra anche i processi degli altri utenti
    - x: mostra i processi non controllati dal terminale
    - e: mostra tutti i processi
  - A seconda delle opzioni alcune informazioni riportate sui processi sono
    - PID numero di identificazione del processo
    - TTY codice di identificazione del terminale
    - STAT stato del processo
    - TIME : istante di tempo in cui è stato fatto partire il processo
    - CMD: comando

Proprietà e Permessi

- Effettuato il login, la shell è dotata di un’identità
  - un real user ID (UID)
  - Uno o più real group ID (GID)
- E’ possibile usare il comando id per conoscere tali valori

Permessi

- Ogni file UNIX/Linux ha un insieme di permessi ad esso associati per controllare chi può *leggere* (*r*), *scrivere* (*w*) o *eseguire* (x) il file
- Un totale di nove bit costituisce i permessi di un file
  - Il primo insieme di tre bit controlla i permessi del proprietario del file
  - Il secondo insieme di tre bit controlla i permessi del gruppo
  - L'ultimo insieme di tre bit controlla i permessi per tutti gli altri

```
ls –l prova.c
-rw-r--r–- giusal giusal 47 2009-05-12 16:32 prova
```

- L'utente deve essere il proprietario dei file per cambiarne i permessi

Permessi (2)

- Modalità dei permessi

Ottale	Binario	Permessi
0	000	Nessuno
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

Maschera di creazione dei file

- Quando è creato un file ad esso sono assegnati dei permessi di default
- I permessi sono determinati dal programma che crea il file
- I processi figli ereditano la maschera di default dai processi genitori
- L'utente può cambiare la maschera per la shell usando il comando umask o impostandolo nei file di inizializzazione della shell
- Il comando umask è usato per rimuovere i permessi dalla maschera esistente
- Inizialmente, umask è 000
  - Corrisponde a dare i permessi 777 (rwxrwxrwx) alle directory e 666 (rw-rw-rw-) ai file per default
  - Per molti sistemi ad umask è assegnato il valore 022 dal programma /bin/login o dal file di inizializzazione /etc/profile

Maschera di creazione file

Esempio: il valore di umask è sottratto dalle impostazioni di default sia per i permessi delle directory che dei file

777 (Directory)

-022 (valore di umask)

755

Risultato: drwxr-xr-x

666 (File)

-022 (valore di umask)

644

Risultato: -rw-r--r--

Cambiare permessi e proprietà

- Il comando chmod
  - Cambia i permessi di file e directory
  - C’è un unico proprietario per ogni file e solo il proprietario (o il superutente) può cambiare i permessi di file e directory
  - Un gruppo può avere un numero di membri, il proprietario del file può cambiare i permessi del gruppo su un file in modo che il gruppo possa beneficiare dei privilegi speciali
  - Per vedere i permessi di un file, dal prompt della shell si può digitare ls –l nomefile

Cambiare permessi e proprietà (2)

- Esempi:

```
$ chmod 755 file
$ ls –l file
-rwxr-xr-x 1 giusal  0 Mar 7 12:52 file
$ chmod g+w file
$ ls –l file
-rwxrwxr-x  1 giusal 0 Mar 7 12:54 file
$ chmod go-rx file
$ ls –l file
-rwx-w----  1 giusal 0 Mar 7 12:56 file
$ chmod a=r file
$ ls –l file
-r--r--r--  1 giusal 0 Mar 7 12:58 file
```

Sintassi di chmod

- La sintassi è la seguente:
  - chmod [opzione..] XYZ nome_file
    - X:indica gli utenti nei confronti dei quali si vuole modificare l'accesso
      - a:tutti gli utenti
      - g: tutti gli utenti del certo gruppo
      - u: l'utente stesso(da user)
      - o: tutti gli altri utenti
    - Y: indica che azione si vuole compiere
      - +:attivare
      - - :togliere
    - Z:indica che tipo di permessi si vuole modificare
      - r: lettura
      - w:scrittura
      - x:esecuzione (da execute)

Cambiare permessi e proprietà (3)

- Il comando chown
  - Cambia il proprietario ed il gruppo dei file e delle directory
  - Su sistemi Linux solo il superutente (root) può cambiare la proprietà
  - Su sistemi UNIX il proprietario del file o il superutente può cambiare la proprietà

Cambiare permessi e proprietà (5)

Esempi (Il file filetest è di proprietà dell'utente e del gruppo giusal)

```
$ ls –l filetest
-rw-rw-r--  1 giusal giusal   0 Jan 12:19 filetest
$ chown root filetest
Chown: filetest: Operation not permitted
$ su root
Password:
# ls –l filetest
-rw-rw-r--  1 giusal giusal     0 Jan 12:19 filetest
# chown root filetest
# ls –l filetest
-rw-rw-r--  1 root giusal     0 Jan 12:19 filetest
# chown root:root filetest
# ls –l filetest
-rw-rw-r--  1 root root     0 Jan 12:19 filetest
```

Il comando su

- Il comando su (significa set user) seleziona l'utente specificato oppure, se utilizzato senza parametri, seleziona l'amministratore di sistema
- ESEMPI:
  - su mario … diviene l'utente mario
  - su root … diviene amministratore
  - su … diviene amministratore (è implicito ROOT)
- A seguito del comando su viene sempre chiesto di inserire la password relativa all'utente scelto

La directory di lavoro

- Ad un utente che ha effettuato il login è assegnata una directory di lavoro nel filesystem chiamata home directory
  - La directory di lavoro è ereditata dai processi generati dalla shell
  - Un qualsiasi processo figlio può cambiare la propria directory di lavoro tuttavia tale cambiamento non avrà effetto sul processo shell padre
  - Il comando cd, usato per cambiare la directory di lavoro, è un comando built-in di shell
    - Ogni shell ha la propria copia del comando cd: Un comando built-in è eseguito direttamente dalla shell come parte del codice della shell
      - La shell non esegue chiamate di sistema fork o exec per eseguire comandi built-in
    - Se un'altra shell (script) è generata dal processo shell padre, ed è eseguito un comando cd dalla shell figlio, la directory cambierà solo nella shell figlio
      - Quando il figlio termina, la shell padre sarà nella stessa directory in cui si trovava prima dell'avvio del figlio

La directory di lavoro (2)

- Esempio

```
$ cd /
$ pwd
/
$ bash
$ cd /home
$ pwd
/home
$ exit
$ pwd
/
$
```

Variabili

- La shell può definire due tipi di variabili: locali e di ambiente
- Le variabili contengono informazioni usate per personalizzare la shell e informazioni richieste da altri processi in modo da poter funzionare
- Le variabili locali sono private alla shell in cui sono create e non sono passate ad alcun processo generato da quella shell
- Le variabili di ambiente, invece, sono passate dai processi padri ai processi figli, dai figli ai nipoti ecc.
  - Alcune variabili di ambiente sono ereditate dalla shell di login dal programma /bin/login
  - Altre sono create nei file di inizializzazione dell'utente, negli script o da riga di comando
- Se una variabile di ambiente è impostata in una shell figlio, non sarà passata indietro alla shell padre
- Il comando built-in set mostra variabili locali e di ambiente
- Il comando env visualizza le variabili di ambiente

Redirezione

- Tutto l'I/O, inclusi file, pipe e socket, è gestito dal kernel mediante un meccanismo chiamato descrittore di file (file descriptor)
  - E' un intero piccolo senza segno, un indice in una tabella dei descrittori di file mantenuta dal kernel e da esso usata per il riferimento ai file aperti e agli stream di I/O
- Ogni processo figlio eredita dal padre la propria tabella dei descrittori di file
  - I primi tre descrittori di file, 0, 1 e 2 sono assegnati al terminale
    - 0 è lo standard input, 1 lo standard output e 2 lo standard error
  - Quando si apre un file, il primo descrittore disponibile è il 3 e sarà assegnato al nuovo file

Redirezione (2)

- Quando un descrittore di file è assegnato ad altro che non sia il terminale, è chiamato redirezione di I/O
- La shell esegue la redirezione dell'output verso un file chiudendo il descrittore del file associato allo standard output, 1 (il terminale), ed assegnando successivamente quel descrittore al file
- Quando redirige lo standard input, la shell chiude il descrittore di file 0 (il terminale) ed assegna quel descrittore ad un file

Redirezione (3)

- Esempio

```
$ who > file
```

L’output del comando who è rediretto dal terminale a file

```
$ cat file1 file2 >> file3
```

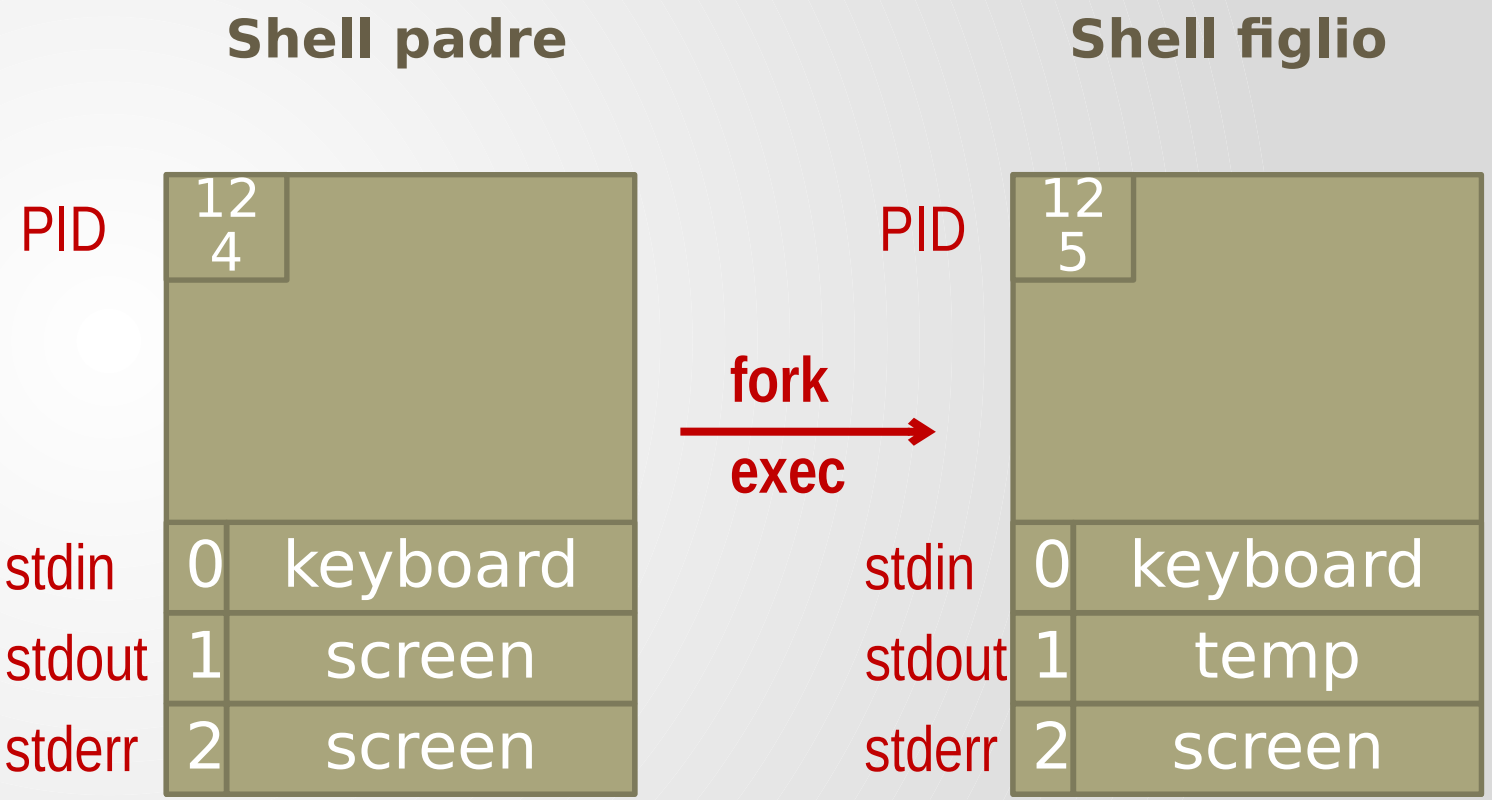
L’output del comando cat (concatena file1 e file2) è aggiunto alla fine del file3

```
$ find / -name file –print 2> errors
```

Un qualsiasi errore dal comando find è rediretto nel file errors. L’output va al terminale

Redirezione (4)

```
grep Tom datafile > temp
```



Pipe

- Le pipe costituiscono la prima forma di interprocess communication (IPC) di UNIX
  - Una pipe consente ai processi di comunicare tra di loro
  - E' un meccanismo per cui l'output di un comando è inviato in input ad un altro comando
    - Il flusso dei dati può avvenire in una sola direzione
  - La shell implementa le pipe chiudendo ed aprendo i descrittori di file
    - Tuttavia, invece di assegnare i descrittori ad un file, li assegna ad un descrittore di pipe (creato con la chiamata di sistema pipe)
- Per semplificare, una pipe non è altro che un buffer del kernel da cui ambo i processi possono condividere i dati eliminando, così, la necessità di file temporanei intermedi
  - L'output di un comando è inviato al buffer e quando il buffer è pieno o il comando ha terminato, il comando sulla destra dell'operatore di pipe legge dal buffer
  - Il kernel sincronizza le attività in modo che un processo attende mentre l'altro legge dal o scrive nel buffer

Pipe (2)

- Esempio

```
$ who | wc
```

Volendo svolgere lo stesso compito senza la pipe, dovremmo usare tre passi

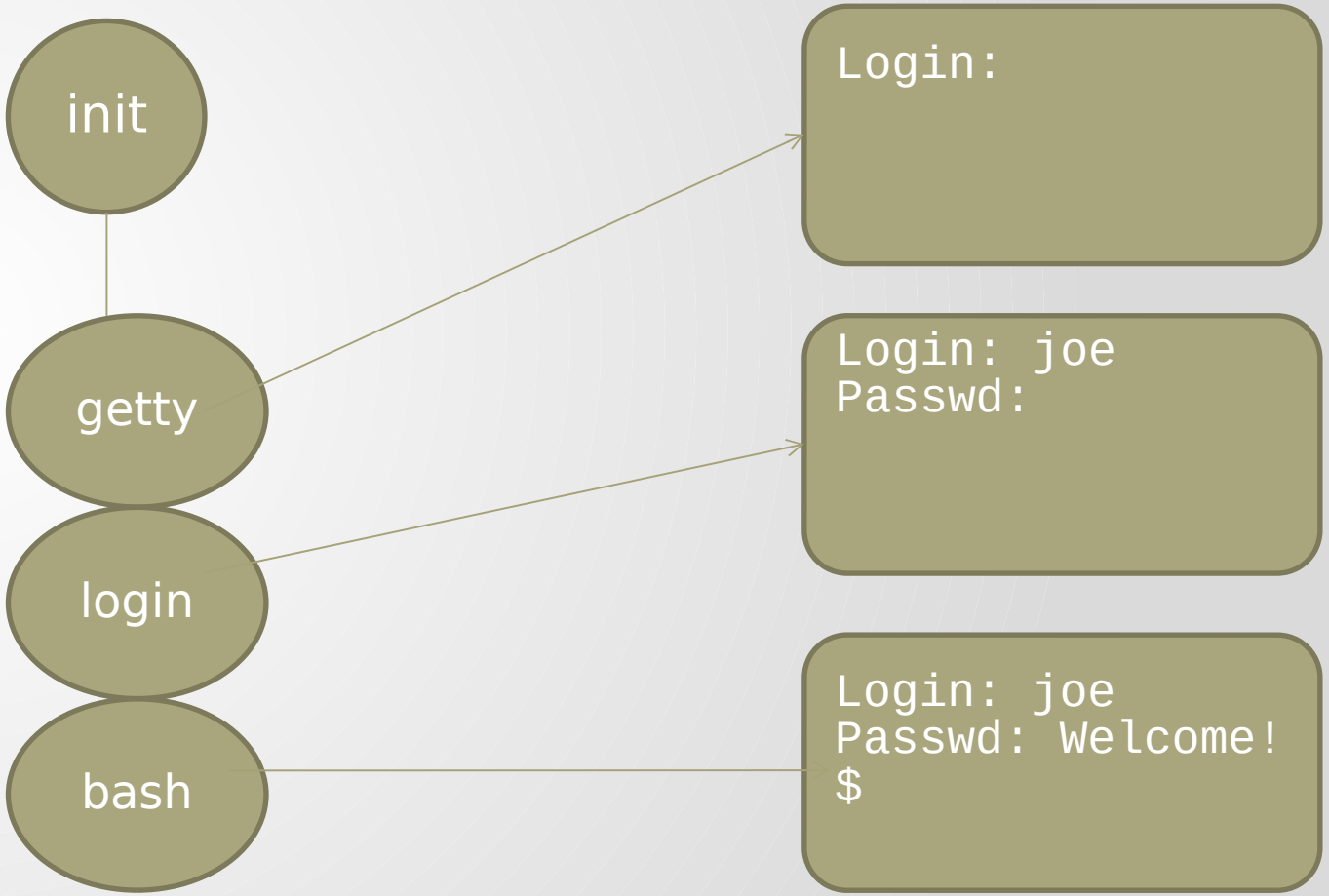
```
$ who > tempfile
$ wc tempfile
$ rm tempfile
```

La Bourne Again shell (Bash)

Startup

- Quando il sistema esegue il boot il primo processo a partire è init con PID = 1
  - Esso genera un processo getty
    - getty apre le porte del terminale definendo i posti da cui proviene lo standard input e in cui vanno gli standard output ed error. Inoltre pone sullo schermo un prompt di login
    - E' eseguito poi il programma /bin/login che richiede una password, crittografa e verifica la password, imposta un ambiente iniziale ed avvia la shell di login /bin/bash
    - Il processo bash cerca il file di sistema /etc/profile ed esegue i suoi comandi
      - Cerca, poi, nella directory home il file di inizializzazione .bash_profile e ne esegue i comandi
      - Dopo esegue un comando dal file di ambiente, solitamente .bashrc ed infine appare il prompt di default ($) sullo schermo

Startup


init
getty
login
bash
Login:
Login: joe
Passwd:
Login: joe
Passwd: Welcome!
$

Impostazione delle opzioni della bash: comando built-in **set**

- Le opzioni set –o
  - Il comando prende opzioni quando è usato –o
    - Le opzioni consentono di personalizzare l'ambiente della shell. Possono essere attive o disattive e sono impostate normalmente nel file .bashrc
- Il formato del comando è:
  - set –o option attiva l'opzione
  - set +o option disattiva l'opzione
  - set –[a-z] abbreviazione per un'opzione; il simbolo - l'attiva
  - set +[a-z] abbreviazione per un'opzione; il simbolo + la disattiva
- Es.: set –o allexport oppure set –a
  - Attiva l'opzione allexport, le variabili sono esportate nelle subshell

I comandi della shell

history

- Tutti i comandi eseguiti vengono memorizzati all'interno di un file di sistema che si trova all'interno della directory home dell'utente e che si chiama .bash_history (NOTA: è preceduto dal punto perché si tratta di un file nascosto)
- Ad esempio l'utente “mau” avrà un suo file “con lo storico dei comandi” all'interno della directory /home/mau
- Per visualizzare il file occorre considerare che si tratta di un file nascosto che viene visualizzato con “ ls –a”
- Per visualizzare i comandi presenti in questo file è possibile usare il comando history, che equivale a: “cat .bash_history”

Uso dei comandi di history e tasto TAB

- Ad esempio, se digitando history si ottiene:

  1 ls
  2 cd miadir
  3 cat pippo
  4 ls -lai

- digitando il comando: !3 viene eseguito il comando 'cat pippo' (questo comando visualizza il file pippo)
- Un'altra scorciatoia molto utile e' il tasto TAB. E' possibile espandere nomi di file o di comandi digitando una serie di caratteri (pattern) e premendo il tasto TAB
- Ad esempio, supponiamo di voler eseguire il comando “touch”, ma non ricordiamo esattamente il nome del comando. Sappiamo solo che inizia per ‘t' o per ‘to'
  - Digitando “t” e premendo TAB, il sistema visualizzerà tutti i comandi che iniziano per “t”

Un po’ di comandi

- I principali comandi per ottenere delle informazioni sullo stato del proprio sistema Linux sono i seguenti:
  - date: Indica la data e l'ora del sistema
  - who: Indica gli utilizzatori connessi al sistema
  - man: Visualizza la documentazione integrata(per uscire si usa il tasto “q”)
- Il comando man permette d'ottenere delle informazioni su comandi. Si tratta di una documentazione in linea in cui si trova:

  NAME: Nome del comando con breve descrizione
  SYNOPSIS: Possibilità di scelta del comando (la sua sintassi)
  DESCRIPTION: Spiegazione corta e condensata delle conseguenze del comando
  FILES: Indicazione dei file modificati dal comando o necessari alla sua esecuzione
  DIAGNOSTICS: Spiegazione degli eventuali messaggi d'errore
  BUGS: Errori conosciuti e ripetuti
  EXAMPLE: Esempi basati sul comando

Il comando man

- Il manuale è organizzato in sezioni e sottosezioni; ogni sezione è composta di pagine (logiche)
- Ogni pagina descrive un singolo argomento (es.: un comando)

Sezione	Contenuti
1	Commands
2	System calls
3	Library functions
4	Special files
5	File formats
6	Games
7	Miscellaneous Information
8	Administration

Il comando man

- man [opzione...] titolo...
  - visualizza le pagine del manuale specificate mediante i suoi parametri
  - -s permette di specificare la sezione

Esempio

```
$ man who
…
$ man –s 2 kill (oppure man 2 kill)
…
$ man –s 2 intro (oppure man 2 intro)
```

- Note
  - Se il numero di sezione non è specificato, è selezionata la prima occorrenza
  - Ogni sezione o sottosezione inizia con una pagina chiamata intro

Visualizzare il contenuto di un file

- E’ possibile usare il comando cat
  - `cat file1 file2`
    - Concatena file1 e file2 ed invia allo standard output il risultato
  - `cat file`
    - Invia il contenuto di file allo standard output
  - `cat`
    - Copia lo standard input sullo standard output

Alias

- Un alias è un’abbreviazione di un comando definita dall’utente della bash
  - Sono utili se un comando ha un lunga serie di opzioni ed argomenti oppure se la sintassi è difficile da ricordare
- Gli alias impostati dalla riga di comando non sono ereditati dalle subshell
- Gli alias di solito sono impostati nel file .bashrc
  - Poiché quando è avviata una nuova shell è eseguito il file .bashrc ogni alias impostato in una shell padre sarà resettato nella nuova shell

Elencare gli alias

- Il comando built-in alias elenca tutti gli alias impostati

```
$ alias
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
```

Creare gli alias

- Il comando alias è usato per creare un alias
  - Il primo argomento è il nome dell'alias
  - La riga rimanente consiste del comando o comandi che saranno eseguiti quando è eseguito l'alias
  - Esempio:

```
$ alias m=more
$ alias mroe=more
$ alias lF='ls –alF'
$ alias r='fc –s'
```

Cancellare gli alias

- Il comando unalias è usato per cancellare un alias
- Per disattivare temporaneamente un alias si precede il nome dell'alias con il backslash \
- Esempio

```
$ unalias mroe
$ \ls
```

Comandi frequenti per la gestione delle directory

Gestione delle directory

- /dev contiene i driver dei dispositivi
- /bin e /usr/bin contiene i comandi Linux standard
- /lib e /usr/lib contiene le librerie standard di Linux
- /var contiene i file di configurazione ed i LOG file
- /etc contiene i file di configurazione di default
- /usr/local/bin contiene comandi che non fanno parte della distribuzione ma sono stati introdotti successivamente
- /opt contiene software commerciale
- /tmp memorizza file temporanei
- /sbin e /usr/sbin contiene i comandi di sistema dell'amministratore (/sbin sta per "safe" bin)

I primi comandi

- La directory principale, detta “root”, è identificata dal simbolo “/”
- Ogni utente ha una directory personale detta “/home” seguita dal suo nome
- whoami per capire come siamo loggati sulla macchina
- ls per avere l’elenco dei file contenuti nella directory corrente
- per spostarsi nella directory Desktop con il comando
  - cd Desktop
- per ritornare nella directory precedente :
  - cd ..

touch

- E’ usato per modificare gli istanti di accesso e modifica di un file
  - touch [option] file
  - Se file non esiste viene creato
- Per cominciare con la gestione file creiamo un file sul Desktop in cui supponiamo di esserci spostati con il comando cd
- Si digiti il comando
  - touch pluto
- Se ora digitiamo il comando ls troveremo nell’elenco il file “pluto”

Copia di un file: cp

- Il comando cp serve per copiare un file da una directory ad un'altra
- La sintassi è del tipo:
  - cp *file_sorgente directory_di_destinazione (o file di destinazione)*
    - *cp pluto Directory* ...Directory è una directory
    - *cp pluto pluto1* ...ora pluto1 è un file
- ESEMPIO:
  - Si crei la directory prova con il comando
    - `mkdir prova`
  - Digitiamo il comando:
    - `cp pluto prova`
- Verifichiamo lo spostamento effettuato

Spostare un file: mv

- Questo comando serve a spostare o a rinominare un file. La sintassi è la seguente:
  - mv pluto pippo ...in tal modo il file di nome pluto ora si chiama pippo
  - mv pluto nome_directory ...sposta il file nella directory indicata eliminando il file della posizione originale
- ESEMPIO: creiamo un'altra directory
  - `mkdir agenda`
  - `mv pluto agenda`
- Vale anche per lo spostamento di una directory intera

Cancellare un file: rm

- E’ il comando utile alla rimozione dei file o directory. La sintassi è la seguente:
  - rm pluto (rimuove il file indicato)
  - rm –r agenda(rimuove una directory con tutto il contenuto)

Cercare file: find

- Questo comando è utile per ricercare file e directory in base al nome immesso, alla data di creazione o alla sua dimensione (espressa in KB)
- La sintassi è:
  - *find* directory_ove_cercare –option expression
  - Esempio:
    - `find . –name pluto` ...trova il file pluto nella directory corrente
    - `find . –ctime 7` ...trova i file che sono stati creati da 7 giorni
    - `find . –size nKB` ...trova tutti i file grandi più di nKB

sort

- Legge dei dati linea per linea dallo standard input e fornisce i dati in ordine alfabetico
- La sintassi è:
  - sort nomefile
  - sort (opzione) –r nomefile
- ESEMPIO:
  - `sort elenco.txt`
  - `sort –r elenco.txt`

Inviare segnali ad un processo

kill

- Tale comando occorre per mandare un segnale ad un processo identificato con il PID (process id). La sintassi è la seguente:
  - kill -s tipo_segnale PID
- Il segnale viene definito attraverso un nome, ad esempio SIGKILL, o un numero di segnale. Il nome del segnale non è sensibile alla differenza tra maiuscole e minuscole e può essere indicato anche senza il prefisso SIG. Se non viene indicato il tipo di segnale da inviare, si intende SIGTERM
- ESEMPI:
  - `kill –l`

  elenca tutta la scelta dei possibili segnali in una tabella
  - `kill –s stop PID_number`

  invia un segnale di stop al processo identificato con quel numero
  - `kill –s cont PID_number`

  riprende il processo fermato (digitarlo da altra shell)

Processi foreground

- Tutti i processi avviati da shell sono eseguiti in foreground, ovvero assumono il controllo del terminale
- Tuttavia è possibile intervenire sulla modalità di esecuzione
  - ctrl+Z: sospende un processo foreground in esecuzione
  - fg: ripristina in foreground un processo sospeso
  - bg ripristina in background un processo sospeso
  - jobs: mostra tutti i processi in background

- Come esempio, basta eseguire il comando gedit (editor di testo), digitare ctrl+Z per sospenderlo ed infine digitare fg per ripristinarlo in foreground

Processi background

- Se digitiamo un comando seguito dal simbolo &, il comando sarà eseguito in background e il controllo sarà subito restituito alla Shell, senza attendere che il comando termini

- La sintassi è la seguente:
  - comando argomenti &

- Esempio
  - gedit &

Esempio background

- $ gedit &
- (gedit:31210): ...si avvia GEDIT
- digitiamo il comando jobs e si vede che è RUNNING in BACKGROUND
  - [1] Stopped man createuser (wd: /home/mario)
  - [2] Stopped more = pluto (wd: /home/mario)
  - [3] Stopped man more (wd: /home/mario)
  - [4] Stopped more = pluto (wd: /home/mario)
  - [5] Stopped more = pluto (wd: /home/mario)
  - [6] Stopped more '=' pluto (wd: /home/mario)
  - [7] Stopped man more (wd: /home/mario)
  - [8] Stopped more '=' pluto (wd: /mario)
  - [9] Stopped more elenco.txt
  - [10] Stopped more -2 elenco.txt
  - [11]- Stopped man sort
  - [12]+ Stopped vim
  - [13] Running gedit &
- $

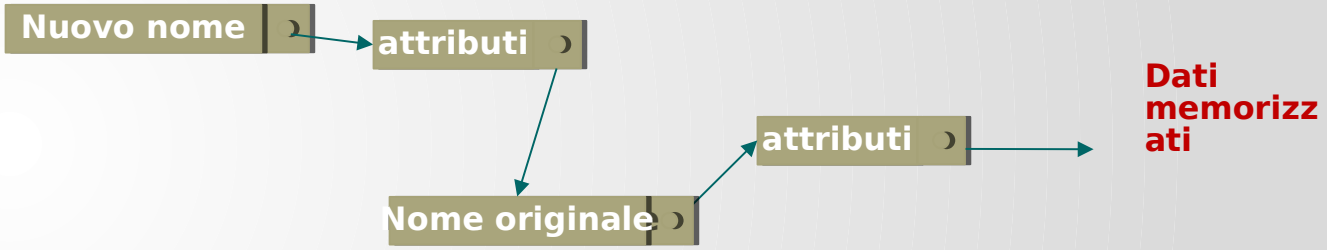
I link

Link

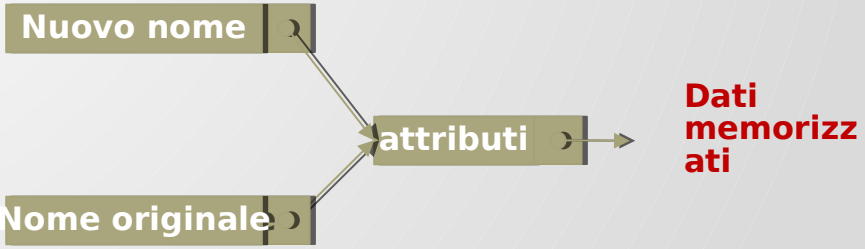
- Sono scorciatoie per accedere a file o directory
  - usate frequentemente
  - condivise
  - usate attraverso nomi diversi
- Permettono di avere più di un punto di accesso per lo stesso file o directory
  - le informazioni rimangono in una unica copia

Tipologie di link

- Due tipi di link:
  - soft link: simili ai collegamenti di windows il file creato contiene solo il nome del riferimento



  - hard link: non viene creato alcun file, si fa riferimento agli stessi dati fisici



  - I due nomi diventano equivalenti

Hard link

- Un link permette di accedere ad un file utilizzando un nuovo *path name*
- Se utilizzo un hard link i due file fanno riferimento allo stesso *i-node*
  - Il numero di hard link che fanno riferimento ad un i-node è memorizzato in un apposito campo dell'i-node stesso (count)
  - Quando un utente cancella un link al file, il contatore viene decrementato, mentre l'i-node viene cancellato solo quando il contatore va a zero
  - Questa soluzione ha il pregio di risparmiare memoria e di permettere accessi veloci ai file collegati ma può generare situazioni anomale in cui un file continua ad essere presente nel FS anche dopo che l'owner effettivo lo ha rimosso con successo

Soft link

- Un link simbolico corrisponde invece ad un file speciale (link) che contiene il path name del file collegato
  - Quando accediamo al file attraverso il link simbolico il pathname viene seguito per recuperare le informazioni relative al file
  - In questo caso la cancellazione del link simbolico non modifica il file originale. La cancellazione del file da parte dell'owner automaticamente rende il file inaccessibile a tutti i link simbolici ad esso collegati
  - Lo svantaggio di questa tecnica è il maggior spreco di memoria e la maggior lentezza nell'accesso

Effetto uso dei link

- nuovo è un soft link a vecchio
  - posso usare indifferentemente nuovo o vecchio per modificare i miei dati
  - se cancello vecchio perdo i miei dati, nuovo rimane, ma nel momento in cui lo si utilizza si ottiene un errore (è un puntatore non inizializzato correttamente) (dangling pointer)
  - se cancello nuovo perdo solo un modo di accedere ai dati

- nuovo è un hard link a vecchio
  - posso usare indifferentemente nuovo o vecchio per modificare i miei dati
  - se cancello vecchio posso ancora accedere ai dati tramite nuovo
  - se cancello nuovo posso ancora accedere ai dati tramite vecchio
  - se cancello vecchio e poi lo ricreo ho ora due insiemi di dati diversi

Esempio

```
$ touch miofile
$ ln miofile miofile-hard
$ ln -s miofile miofile-soft
$ ls -il miofile*
totale 3
50209428 -rw-rw-r--    2 giusal giusal 418  Mar 22 12:00 miofile
50209428 -rw-rw-r--    2 giusal giusal  418 Mar 22 12:00 miofile-hard
21331971 lrwxrwxrwx 1 giusal giusal      7 Mar 22 12:01 miofile-soft -> miofile
```

miofile e miofile-hard hanno lo stesso i-node (prima colonna), mentre miofile-soft ne ha uno diverso

I link simbolici hanno la lettera "l" all'inizio dei permessi. Alla fine viene anche indicato dove punta il collegamento: (-> miofile).

Nei link simbolici, vengono mostrati come attivi tutti i permessi di lettura, scrittura ed esecuzione per tutti gli utenti, perché quelli che contano sono in realtà i permessi del file (o della directory) cui effettivamente punta il collegamento. Ciò non avviene con gli hard link

Esempio

```
$ echo "Hello world" > file1
$ ln file1 file2
$ ls -li
totale 8
21331972 -rw-rw-r--    2 staiano  staiano    12 Mar  22  12:05 file1
21331972 -rw-rw-r--    2 staiano  staiano    12 Mar  22  12:05 file2
$ cat file2
Hello world
$ cat file1
Hello world
$ rm file1
$ cat file2
Hello world
$ ls -li
totale 4
21331972 -rw-rw-r--    1 giusal  giusal     12 Mar 22  12:05 file2
```

Quando si cancella un file, Linux elimina il riferimento al suo i-node dalla directory che lo contiene.

Quindi quando si utilizzano i collegamenti fisici, un file viene cancellato effettivamente solo quando sono stati eliminati tutti i riferimenti a questo.

Esempio

```
$ ln -s file2 file1
$ ls -li
totale 4
21331973 lrwxrwxrwx 1 giusal giusal  5 Mar 22 12:13 file1 -> file2
21331972 -rw-rw-r-- 1 giusal giusal 12 Mar 22 12:05 file2
$ cat file1
Hello world
$ cat file2
Hello world
$ rm file2
$ cat file1
file1: No such file or directory
$ ls -la
totale 8
drwxrwxr-x  2 giusal  giusal 4096 apr  1 00:03 .
drwx------ 23 giusal  giusal 4096 mar 31 23:35 ..
lrwxrwxrwx  1 giusal  giusal    5  mar 22 12:13 file1 -> file2
```

Cancellando il file a cui fa riferimento un link simbolico quest'ultimo ha nel suo i-node tutti gli attributi del file ma non punta più ad alcuna area di memoria, producendo la segnalazione di errore.