Lezione 5 - L'shell Bash e l'X Windows System

3 dicembre 2006

Alessio Checcucci

Riccardo Aldinucci

Q.It Universita' degli Studi di Siena

1 La configurazione della Bash shell

1.1 Personalizzare il proprio environment

Un *environment* e' costituito da un insieme di concetti che esprimono le cose che un computer fa' in termini comprensibili e coerenti e da un *look and feel* che sia confortevole.

Le shell UNIX presentano all'utente delle entita' come file, directory, standard I/O, ecc., mentre e' UNIX stesso che permette di interagire con queste entita'. Il *look and feel* del nostro environment e' determinato dalla tastiera e dal display, ma anche da una serie di altri parametri, che comportano un livello di sofisticazione nella personalizzazione piuttosto elevato.

Le quattro piu' importanti caratteristiche che Bash fornisce per la personalizzazione dell'environment sono:

- *Special files*
  I file *.bash_profile, .bashrc e .bash_logout* che la shell legge quando viene fatto il login ed il logout da una nuova shell.

- *Aliases*
  Sinonimi per comandi o stringhe di comandi definibili per comodita'.

- *Options*
  Controllo di vari aspetti dell'environment, possono essere attivate e disattivate.

- *Variables*
  Valori variabili referenziati tramite un nome. La shell e altri programmi possono modificare il loro comportamento in base al valore che contengono le variabili.

Sebbene queste non siano le sole possibilita', sono peraltro la base per una personalizzazione avanzata e costituiscono anche un aspetto comune di tutte le shell UNIX.

1.2 I file .bash_profile, .bashrc e .bash_logout

Questi sono tre file che si trovano nella home directory dell'utente e che hanno un significato particolare per bash. Essi forniscono un modo per personalizzare l'environment automaticamente al login, oppure quando viene invocata un'altra bash shell e per eseguire dei comandi al logout.

In genere questi file esistono gia' nella home directory perche' sono stati copiati alla creazione dell'utente dalla directory /etc/skel. Nel caso non esistessero l'account utilizzera' solo il file di default */etc/profile*. Questi file sono normali plain ASCII file che possono essere modificati con qualsiasi editor.

Il file piu' importante e' *.bash_profile*, esso viene letto e i comandi che contiene sono eseguiti da bash ogni volta che un utente fa login nel sistema.

Vale la pena di notare che tutto cio' che viene modificato in .bash_profile non avra' alcun effetto finche' il file non sara' stato letto di nuovo facendo logout e nuovamente login. Alternativamente un utente puo' usare il comando *source* oppure il carattere *"."*:

source .bash_profile oppure *. .bash_profile*

source esegue i comandi nel file che viene specificato. Bash ammette due sinonimi per il file .bash_profile: .bash_login e .profile. Solo uno di questi file viene letto al login nell'ordine di preferenza: .bash_profile, .bash_login, .profile.

Il file .bash_profile viene letto solo dalla login shell, se una subshell viene avviata (per esempio con il comando bash) essa cerchera' di leggere i comandi dal file *.bashrc*. Questo approccio permette, flessibilmente, di separare i comandi necessari al login da quelli di cui si puo' aver bisogno quando si esegue una subshell. Se i comandi di cui si ha bisogno sono gli stessi allora e' suffiente invocare tramite source il file .bashrc dall'interno .bash_profile.

Il file *.bash_logout* viene letto ed eseguito ogni volta che una login shell termina. Viene fornito per poter eseguire comandi all'uscita dal sistema. Molto spesso questo file non esiste e quindi nessun comando verra' eseguito.

1.3 Alias

Ogni utente di UNIX sa che nel sistema ci sono comandi che possono avere nomi criptici, oppure ci sono comandi che vengono utilizzati molto spesso con una serie di opzioni. Gli alias permettono di rinominare questi comandi in modo semplice.

Gli alias possono essere definiti sulla linea di comando, e nei file .bash_profile e .bashrc, utilizzando l'espressione:

alias name=command

Il *name* viene ad essere in questo modo un alias per *command*. A questo punto quando viene invocato *name* come comando, bash sostituira' *command* al suo posto quando va ad eseguire la linea. Vale la pena di notare che **non** ci devono essere spazi prima e dopo il simbolo =. Se command e' costituito da piu' di una parola, allora sara' necessario fare il quoting della stringa.

Occorre sempre ricordare che bash quando incontra un alias fa prima di tutto una sostituzione letterale con command e poi passa all'analisi della riga, in questo modo tutti gli wildcard saranno perfettamente funzionanti.

Come seconda cosa da tenere in mente e' che gli alias sono ricorsivi, cioe' e' possibile definire un alias di un alias. Naturalmente in questo modo sarebbe possibile creare un loop. Ma bash assicura che questo non accade, perche' solo la prima parola del testo che viene sostituito viene controllata per un ulteriore alias, se essa e' identica all'alias che e' stato espanso, esso non viene espanso una seconda volta.

Un alias puo' essere definito solo per l'inizio di una command string, anche se esistono alcune estensioni non documentate che raramente sono utilizzate.

Se il built-in comand *alias* viene lanciato senza argomenti, esso mostrera' tutti gli alias. La sintassi *alias name* al contrario mostrera' il comando relativo a quell'alias. Il comando *unalias name* rimuovera' la definizione dell'alias.

Gli alias sono molto comodi per la personalizzazione dell'ambiente, anche se sono stati superati dall'uso degli shell script e delle funzioni.

1.4 Options

Le *options* permettono di cambiare il comportamento della shell. Le options costituiscono un set in cui ogni valore puo' essere on oppure off. Le options sono moltissime e riguardano gli aspetti piu' disparati della shell. Qui verranno introdotte solo quelle che l'utente utilizzera' di piu':

I comandi base che hanno a che fare con le options sono:
set -o optionname e *set +o optionname*
E' possibile cambiare piu' di un'opzione alla volta con il comando set, il segno - attiva la option, mentre il segno + la disattiva, al contrario di quello che si potrebbe supporre.

La maggior parte delle option hanno associata una lettera che permette di attivarle e disattivarle senza utilizzare la sintassi completa, ma con un comune comando UNIX (per esempio set -o noglob e' l'equivalente di set -f). Comunque l'uso di questa vecchia sintassi viene scoraggiato.

Le opzioni base sono:

Option	**Description**
emacs	Entra nella modalita' di editing emacs
ignoreeof	Non permette di fare logout con un semplice CTRL-D
noclobber	Non permette la redirezione dell'output (>) per sovrascrivere un file esistente
noglob	Non espande gli wildcard per il filename come * e ?
nounset	Indica un errore quando si cerca di usare una variabile che non e' stata definita
vi	Entra nella modalita' di editing vi

Vale la pena di ricordare che esiste un'opzione per accedere alla *restricted shell*: set –restricted o set -r. Questa modalita' e' stata progettata per mettere l'utente (o uno script) in un ambiente in cui la propria capacita di muoversi e di scrivere sia molto limitata.

Shopt Shopt e' un nuovo built-in introdotto in bash versione 2.0 per la configurazione del comportamento della shell. Il costrutto *shopt -o* supporta buona parte delle opzioni di set, e viene fornito per compatibilita'.

Il formato del comando shopt e':
shopt options option-name
Dove options sono:

Option	**Meaning**
-p	Mostra una lista delle variabili impostabili e i loro valori
-s	Set di ogni *option-name*
-u	Unset di ogni *option-name*
-q	Sopprime il normale output
-o	Permette di usare la sintassi del comando set

L'azione di default e' quella di fare l'unset delle variabili.
Una lista dei principali option-names e' la seguente:

Option	**Meaning**
cdable_vars	Se settato, un argomento passato al built-in cd che non sia una directory viene interpretato come un nome di variabile che contiene la directory in cui spostarsi
checkhash	Se settato, bash controlla che un comando esista nella hash table prima di eseguirlo, altrimenti viene fatta una normale path search
cmdlist	Se settato, bash cerca di salvare tutte le linee di un comando multilinea nella stessa history entry
dotglob	Se settato, bash include i file che iniziano con un . (dot) nei risultati della pathname espansion
execfail	Se settato, una shell non interativa non uscira' se non puo' eseguire il file specificato dal comando exec. Una shell interattiva non esce se exec fallisce.
histappend	Se settato, la history list viene appesa al file specificato dalla variabile HISTFILE quando la shell esce, invece di sovrascriverlo.
lithist	Se settato, e cmdhist e' attivo i comandi multilinea vengono salvati nella history con dei newline invece che con il separatore ;
mailwarn	Se settato, bash guardera' la posta a cui si e' avuto accesso dall'ultima volta che e' stata controllata.

1.5 Shell Variables

Ci sono varie caratteristiche dell'ambiente che si possono voler personalizzare, ma che non possono essere espresse con dei valori on/off. Le caratteristiche di questo tipo possono essere specificate con le shell variables.

La shell variable e' un nome che ha un valore associato. Bash ha una serie di variabili built-in a cui possono aggiungersi quelle personali. Per convenzione le variabili sono caratterizzate da un nome tutto maiuscolo.

La definizione di una variabile viene fatta con:

varname=value

Non ci deve essere spazio ai lati dell'= e se ci sono piu' parole e' necessario il quoting.

Per utilizzare il valore di una variabile in un comando e' necessario precederne il nome con un carattere *$*.

E' possibile cancellare una variabile con il comando:

unset varname

Tutte le variabili inesistenti hanno un valore null, ovvero quello di una stringa vuota *""*.

Il modo piu' pratico per controllare il valore di una variabile e' quello di usare il comando *echo*, esso non fara' altro che stampare i propri argomenti.

Variables and quoting Come sappiamo nessuno dei caratteri speciali viene interpretato all'interno delle single quotes, mentre alcuni, e in maniera speciale il *$*, lo sono all'interno delle double quotes. In questo senso le variabili vengono valutate. Le double quotes evitano il normale processo della shell che tende a separare gli argomenti in singole parole, facendogli credere che la stringa sia una sola parola.

Built-in Variables Esistono tutta una serie di built-in variables che sono di interesse per il normale utente UNIX. Le piu' utili sono riportate di seguito:

Editing mode variables

Variable	Meaning
HISTCMD	Il numero nella history dell'attuale comando
HISTCONTROL	Se impostato al valore *ignorespace* le linee che iniziano con spazio non vengono messe nella history. Se il valore e' *ignoredups* le linee uguali all'ultima nella history non vengono inserite. Se il valore e' *ignoreboth* entrambe sono attive
HISTIGNORE	Una lista di pattern separati da *:* per decidere quali comandi vanno nella history.
HISTFILE	Nome dell'history file. Il default e' ~/.bash_history
HISTFILESIZE	Il massimo numero di linee dell'history file
HISTSIZE	Il massimo numero di comandi da memorizzare nella history
FCEDIT	Pathname dell'editor utilizzato dal built-in fc

Prompting Variables Il prompt della shell e' un oggetto altamente configurabile. Per questo e' possibile impostare quattro stringhe per il prompt. Esse vengono memorizzate nelle variabili PS1, PS2, PS3 e PS4. La prima di esse e' la primary shell prompt string ed e' quella che controlla l'aspetto del normale shell prompt utente.

La personalizzazione del prompt per mezzo della variabile PS1 avviene specificando un pattern in cui sono presenti una serie di caratteri di controllo e caratteri letterali:

Command	**Meaning**
\a	Il carattere ASCII *bell*
\d	La data: “weekday month day“
\e	Il carattere ASCII *escape*
\H	L’hostname
\h	L’hostname fino al primo .
\n	Un CR e un LF
\s	Il nome della shell
\T	L’ora corrente in formato HH:MM:SS 12 ore
\t	L’ora corrente in formato HH:MM:SS
\@	L’ora corrente in formato am/pm 12 ore
\u	Lo user name dell’utente corrente
\v	La vesione di bash
\V	La release di bash (versione e pathchlevel)
\w	La current working directory
\W	Il basename della current working directory
\#	Il command number dell’attuale comando
\!	L’history number dell’attuale comando
\$	Se l’UID effettivo e’ 0 stampa un #, altrimenti un *$*
\nnn	Un carattere in codice ottale
\\	Stampa un backslash
\[	Inizia una sequenza di caratteri non stampabili (di controllo del terminale per es.)
\]	Termina una sequenza di caratteri non stampabili

PS2 viene detto il secondary prompt, il suo valore di default e’ >, viene utilizzato quando viene battuta una linea incompleta e premuto RETURN, come indicazione che il comando deve essere completato.

PS3 e PS4 sono relativi alla programmazione della shell e al debugging.

In particolare PS3 specifica il prompt che viene presentato dalla funzione select.

Command search path Una variabile molto importante e' PATH, che aiuta l'utente a trovare i comandi. I comandi che vengono invocati non sono altro che file eseguibili sulla piattaforma su cui stiamo lavorando. Essi si possono trovare in molte directory, ma in ogni caso non dovrebbe esserci motivo per dover sapere il path completo di dove si trova ogni comando per poterlo lanciare.

E' qui che la variabile PATH dimostra la sua importanza. Il suo valore e' costituito da una lista di directory, in cui la shell cerca ogni volta che un comando viene invocato[1], i nomi delle directory sono separati da *:*.

L'operazione che in genere si compie e' quella di aggiungere qualche directory al PATH nel seguente modo:

PATH=$PATH":new_path"

I comandi sono ricercati nell'ordine in cui le directory sono espresse in PATH, per cui cosi' vengono risolti gli eventuali casi di omonimia.

Il built-in *type* fornisce il pathname del comando fornito come argomento, oppure solo il nome del comando e il suo tipo se esso e' un built-in, un alias o una funzione.

Command hashing In genere cercare i comandi in un PATH lungo e' un'operazione piuttosto dispendiosa. Per velocizzare le cose la bash usa la cosiddetta *hash* table.

Ogni volta che la shell trova un comando nel search path, lo mette nella hash table. Se il comando viene utilizzato di nuovo, bash prima controlla la hash table per vedere se esso e' presente. Se lo fosse, utilizza il path registrato nella tabella ed esegue il comando, altrimenti cerca il comando nel search path.

E' possibile vedere il contenuto della hash table con il comando:

hash

che mostra i comandi e il numero di volte in cui sono stati eseguiti nella current login session.

E' possibile svuotare il contenuto della hash table con il comando:

hash -r

Esiste l'opzione *-p* che permette di introdurre un comando nella tabella anche se non esiste.

L'opzione *hashall* del comando set permette di attivare e disattivare la hash table.

Directory search path and variables CDPATH e' una variabile il cui valore e' una lista di directory separate da *:*, la cui funzione e' quella di aumentare le capacita' del comando cd.

Per default questa variabile non viene impostata, quindi il comando *cd dirname* cerchera' dirname nella directory corrente. Se al contrario CDPATH e' stata impostata, il comando cd avra' tutta una serie di path in cui cercare dirname.

Miscellaneous variables Queste sono variabili che non sono utilizzate per la personalizzazione, ma servono come indicatori di stato e per altri vari scopi:

[1] A meno che il comando non contenga una /, nel qual caso la ricerca non ha luogo

Variable	Meaning
HOME	Nome della login directory
SECONDS	Numero di secondi dall'invocazione della shell
BASH	Pathname dell'istanza della shell che sta girando
BASH_VERSION	Il numero di versione della shell che sta girando
BASH_VERSINFO	Un array di version info sulla shell che sta girando
PWD	Current directory
OLDPWD	Directory precedente all'ultimo comando cd

1.6 Personalizzazione e subprocess

La maggior parte delle variabili viste sono di pertinenza della shell e non sono neppure conosciute al di fuori di essa. Quello che dobbiamo vedere e' cosa conoscono i comandi che nella shell vengono lanciati e che sono conosciuti come subprocess. Se il subprocess e' un programma bash, allora tutto puo' essere propagato.

Environment variables Per default, solo un genere di cose viene riconosciuto da tutti i sottoprocessi, una particolare classe di variabili, dette *environment variables.*

Alcune delle variabili built in (HOME, PATH, PWD) sono variabili d'ambiente.

Ogni variabile puo' diventare una variabile d'ambiente, e' sufficiente definirla e poi esportarla:

export varnames

oppure combinare le due operazioni:

export varname=value

E' possibile anche definire variabili che facciano parte dell'ambiente di un particolare subprocess, facendo precedere il comando dall'assegnazione di variabile:

varname=value command

E' possibile vedere l'elenco delle variabili d'ambiente con il comando:

export -p

Molte variabili sono state talmente utilizzate nella storia di UNIX che sono diventate degli standard, sebbene non siano built-in in bash (a differenza di Korn):

Variable	**Meaning**
COLUMNS	Il numero di colonne del display
EDITOR	Pathname dell'editor da utilizzare
LINES	Il numero di linee del display
SHELL	Pathname della shell che sta girando
TERM	Il tipo di terminale che si sta usando

Terminal Types La variabile TERM in particolare e' di grande importatnza per tutti i programmi che utilizzano l'intero schermo o una finestra. Il valore della variabile TERM deve essere una stringa in lettere minuscole che appaia come nome di file nel *terminfo* database (un database a doppio livello che si puo' trovare in genere sotto /usr/share/terminfo).

Questa directory contiene un primo livello di subdirectory il cui nome e' costituito da un solo carattere. Ognuna di esse contiene a sua volta dei file con le terminal information per i terminali il cui nome inizia con quella lettera. Le informazioni sono in forma binaria e ragguagliano su come pilotare i terminali.

L'environment file Sebbene le variabili d'ambiente vengano sempre passate ai subprocess, alla shell occorre dire quali altre entita' (alias, opzioni, variabili, etc.) debbano conoscere i sottoprocessi.

Il modo per realizzare tutto questo e' quello di mettere tutte le definizioni necessarie nell'*environment file*, che per default in bash e' *.bashrc*.

Come regola generale si dovrebbero mettere la maggior parte delle definizioni nel file .bashrc e far si che il file .bash_profile richiami quest'ultimo con un source. Le sole cose che vanno messe nel file .bash_profile sono variabili d'ambiente e exports che girano o producono output solo al login time.

2 La programmazione della bash shell

2.1 Shell script and functions

Uno *script* che e' un file che contiene comandi di shell, ovvero e' un programma della shell.

E' possibile crearne uno con text editor; per eseguirlo ci sono due metodi. Il primo e' invocarlo con il comando

source filename

l'altro modo e' quello di scriverne semplicemente il nome e premere RETURN, naturalmente lo script deve avere il permesso di esecuzione.

C'e' un'altra differenza fra i due metodi di esecuzione. Con source, i comandi dello script sono eseguiti come se facessero parte della login session. Al contrario nel secondo modo, la shell compie una serie di azioni:

- Lancia un'altra copia della shell come subprocess, detta *subshell*. La subshell prendera' i comandi dallo script, li eseguira' e terminera' restituendo il controllo alla parent shell.

Tutte le variabili esportate dalla parent shell saranno viste dalla subshell.

Functions Una funzione e' una sorta di uno script all'interno di uno script, che viene utilizzato per definire del codice di shell e un nome con cui referenziarlo; esso viene messo nella memoria della shell per essere invocato ed eseguito in seguito.

Le funzioni sono molto efficienti perche' il loro codice e' in memoria e permettono una programmazione estremamente strutturata e modulare.

Una definizione di funzione si puo' fare in due modi:

function *functname*
{
shell commands
}
oppure
functname()

{
shell commands
}

E' possibile cancellare una funzione con in comando *unset -f functname*.

Quando una funzione viene definita, alla shell viene detto di salvare il suo nome e la definizione (i comandi) in memoria. Per eseguire la funzione e' sufficiente scriverne il nome oltre ad eventuali argomenti.

E possibile vedere l'elenco delle funzioni definite per mezzo del comando:
declare -f

Ci sono due differenze principali fra script e funzioni:

- Le funzioni non sono eseguite in una subshell
- Se una funzione ha lo stesso nome di uno script o di un eseguibile, essa ha la precedenza

Ordine di precedenza dei comandi L'ordine di precedenza per le varie sorgenti di comandi e' il seguente:

- Alias
- Parole chiave (keywords)
- Funzioni
- Built-in
- Script e programmi eseguibili

Se fosse necessario sapere la fonte di un comando e' possibile utilizzare il comando:
type -all

2.2 Shell variables

I fulcro di ogni linguaggio di programmazione sono le variabili, bash pone una particolare enfasi sulle stringhe di caratteri, oltre a questo bash fornisce anche dei meccanismi addizionali per operare con gli interi in modo esplicito.

Positional parameters Le piu' importanti variabili built-in sono i parametri posizionali. Sono deputati a contenere gli argomenti della linea di comando quando gli script sono invocati. I parametri posizionali hanno i nomi 1, 2, 3, ecc. e i loro valori sono referenziabili mediante $1, $2, $3, ecc. Esiste anche il parametro posizionale 0, il cui valore e' il nome dello script.

Due variabili speciali contengono tutti i parametri posizionali, @ e *. La loro differenza e' sottile e diviene importante quando vengono espressi dentro le double quotes:

- "$*" e' un'unica stringa che consiste di tutti i parametri posizionali separati dal primo carattere della variabile d'ambiente IFS (internal field separator).

- "$@" e' uguale a "$1", "$2", ..., "$*N*", dove *N* e' il numero dei parametri posizionali. Ovvero e' equivalente a N variabili in double quotes separate da spazi.

La variabile # contiene il numero di parametri posizionali. Tutti i parametri posizionali sono read-only.

Approfondimento sulla sintassi delle variabili Fino ad ora e' stata fatta una semplificazione su come le variabili vengono richiamate. Abbiamo detto che il loro valore puo' essere ottenuto con la sintassi *$varname*, essa e' una forma piu' semplice della sintassi piu' generale *${varname}*.

E' possibile omettere le curly braces ogni volta che il nome della variabile sia seguito da un carattere che non e' una lettera, un numero o l'underscore.

2.3 String Operators

Gli operatori sulle stringhe permettono di manipolare il valore delle variabili in molti modi senza dover scrivere interi programmi o fare affidamento a programmi esterni. In particolare essi permettono di:

- Assicurarsi che le variabili esistano
- Impostare un default value per le variabili
- Catturare gli errori per variabili non impostate
- Estrarre porzioni di varibili che contengano un pattern (pattern matching)

La lista di questi operatori e' molto lunga e complessa, per una trattazione esaustiva si consiglia di riferirsi alla letteratura in appendice.

Syntax of string operators L'idea di base della sintassi degli string operator e' che i caratteri speciali che identificano l'operazione siano inseriti fra il nome della variabile e la *}*. Quasiasi argomento di cui l'operatore possa avere bisogno viene inserito a destra dello stesso.

Alcuni esempi sono:

- *${varname:-word}* - Se *varname* esiste e non e' nulla ritorna il suo valore, altrimenti ritorna *word*
- *${varname:=word}* - Se *varname* esiste e non e' nulla ritorna il suo valore, altrimenti imposta il valore a *word* e ritorna il valore
- *${varname:?message}* - Se *varname* esiste e non e' nulla ritorna il suo valore, altrimenti stampa *varname:* seguito da *message* e termina (abort) il comando o lo script
- *${varname:+word}* - Se *varname* esiste e non e' nulla ritorna word, altrimenti ritorna null
- *${#varname}* - Restituisce la lunghezza del valore della variabile in caratteri

Pattern matching

- *${varname#pattern}* - Se il *pattern* corriponde all'inizio del valore della variabile, cancella la parte piu' breve che corrisponde e ritorna il resto
- *${varname##pattern}* - Se il *pattern* corrisponde all'inizio del valore della variabile, cancella la parte piu' lunga che corrisponde e ritorna il resto
- *${varname%pattern}* - Se il *pattern* corrisponde alla fine del valore della variabile, cancella la parte piu' breve che corrisponde e ritorna il resto
- *${varname%%pattern}* - Se il *pattern* corrisponde alla fine del valore della variabile, cancella la parte piu' lunga che corrisponde e ritorna il resto
- *${varname/pattern/string} ${varname//pattern/string}* - Il piu' lungo match per il *pattern* in *variable* viene sostituito da *string.* Nella prima forma solo il primo match viene sostituito, nella seconda tutti i match sono sostituiti

Facciamo un esempio: supponiamo che la variabile *path* valga */home/alessio/corso/corso.linux.qit*
${path##/*/} -> corso.linux.qit
${path#/*/} -> alessio/corso/corso.linux.qit
${path%.*} -> /home/alesssio/corso/corso.linux
${path%%.*} -> /home/alesssio/corso/corso

2.4 Comand substitution

La sostituzione di comandi permette di utilizzare l'output di un comando come se fosse il valore di una variabile. La sintassi e' la seguente:

$(UNIX command)

Il comando viene eseguito e tutto quello che scrive sullo standard output viene restituito come valore delle espressione. La sostituzione di comandi avviene fra le double quotes, come per l'espansione di variabile e la tilde expansion.

2.5 Flow Control

2.5.1 If/else

Il *conditional* e' il tipo piu' semplice di controllo di flusso, incarnato dall' if statement di bash. La discriminante e' la verita' o la falsita' della *condition.* Le condizioni testano il valore di variabili, caratteristiche di file, exit status dei comandi e altro.

La sintassi e' la seguente:

if *condition*
then
statements
*[**elif** condition*
then *statements ...]*
*[**else***
statements]
fi

La forma piu' semplice esegue gli statement solo se la condizione e' vera. Se viene aggiunta la parola else, allora sara' possibile eseguire gli statement che la seguono se la condizione e' falsa. La forma contratta elif (else if) serve per introdurre ulteriori condizioni, se tutte falliscono allora viene eseguita la parte dopo else.

Exit status e return

- La verita' o la falsita' della condizione viene determinata in base all'*exit status* dei comandi. Ogni comando quando termina restituisce un valore, detto exit status, a chi lo ha invocato (nel nostro caso alla shell). Il valore *0* in genere e' associato ad un OK exit status, mentre qualsiasi altro valore (1-255) denota in genere un errore. Il costrutto *if* controlla l'exit status dell'ultimo statement nella lista che segue la parola chiave if. Se l'exit status e' 0 allora la condizione viene interpretata come vera, ogni altro valore viene interpretato come condizione falsa.

- Lo statement *return N* fa si che la funzione che lo precede esca con un exit status *N*. Il default e' quello di ritornare l'exit status dell'ultimo comando della funzione. Return puo' essere utilizzato solo all'interno delle funzioni e degli shell script eseguiti con source.
  Al contrario lo statement *exit N* esce dallo script, indipendentemente dal punto in cui si trova in una funzione annidata.

- E' possibile combinare logicamente gli exit status, in modo da testare piu' cose per volta. Le sintassi sono:

  - *statement1 && statement2* - E' un AND logico. esegue *statement1* e se l'exit status e' 0, allora esegue *statement2*
  - *statement1 || statement2* - E' un OR logico. esegue *statement1* e se l'exit status non e' 0, allora esegue *statement2*

Condition Tests Gli exit status sono la sola cosa che lo statement if puo' testare. La shell fornisce la possibilita' di fare il test per una serie di condizioni con il costrutto *[...]*.[2]

[condition] e' uno statatement come gli altri, la sola cosa che fa e' restituire un exit status che dice se la *condition* e' vera (gli spazi dopo la prima parentesi e prima dell'ultima sono obbligatori).

String comparison Le square brackets ([]) racchiudono espressioni che includono vari operatori. Se abbiamo due espressioni con un valore stringa *str1* e *str2*:

[2] La sintassi [...] e' in tutto e per tutto un sinonimo per il built-in *test*

Operator	**True if ...**
str1 = str2	str1 e uguale a str2
str1 != str2	str1 e' diversa da str2
str1 < str2	str1 e' lessicograficamente minore di str2
str1 > str2	str1 e' lessicograficamente maggiore di str2
-n str1	str1 non e' null (ha lunghezza > 0)
-z str1	str1 e' null

File attribute checking In questo caso facciamo il check per certi attributi dei file. Esistono venti operatori, ne vedremo alcuni:

Operator	**True if ...**
-d *file*	*file* esiste ed e' una directory
-e *file*	*file* esiste
-f *file*	*file* esiste ed e' un regular file
-r *file*	L'utente ha read permission sul *file*
-s *file*	file esiste e non e' vuoto
-w *file*	L'utente ha write permission sul *file*
-x *file*	L'utente ha execute permission sul *file* o puo' accedere alla directory
-O *file*	L'utente possiede il *file*
-G *file*	Il GID del *file* e' uguale a quello dell'utente
file1 -nt *file2*	Il *file1* e' piu nuovo del *file2*
file1 -ot *file2*	Il *file1* e' piu vecchio del *file2*

Vale la pena di notare come si possano combinare piu' test con gli operatori logici && e ||. Per esempio l'espressione:

if [condition1] && [condition2]; then

E' possibile anche negare il valore di un espressione precedendola con il carattere *!*.

Inoltre e' possibile combinare espressioni logiche complesse all'interno delle parentesi con gli operatori *-a* e *-o*, essi sono disponibili solo all'interno del costrutto test.

Integer conditions La shell fornisce anche la possibilita' di eseguire dei test artimetici. Essi sono distinti da quelli utilizzati per le stringhe.

Test	**Comparison**
-lt	Minore di
-le	Minore o uguale di
-eq	Uguale
-ge	Maggiore o uguale di
-gt	Maggiore di
-ne	Non uguale

2.5.2 for

E' il piu' semplice e piu' utilizzato costrutto per i loop. Esso permettere di ripetere una sezione di codice per un numero prefissato di volte. Durante ogni iterazione una particolare variabile, detta *loop variable*, viene impostata ad un valore diverso; in questo modo ogni iterazione puo' compiere azioni diverse.

Il ciclo for e' simile a quello dei comuni linguaggi. La grossa differenza e' che il for della bash non permette di specificare il numero di iterazioni, invece, permette di specificare una lista di valori. Per questo motivo for e' ideale per lavorare con gli argomenti della linea di comando e con i set di file.

La sintassi del comando for e':

for *name [***in** *list]*
do
statements that use $name
done

List rappresenta una lista di nomi, se list viene omessa, il suo default e' "$@", ovvero la lista degli argomenti della linea di comando in forma quoted.

2.5.3 case

La forma del costrutto *case* in bash e' diversa da quella di altri linguaggi, infatti esso permette di fare il test di stringhe rispetto a dei pattern, che possono contenere dei caratteri wildcard. Case permette di esprimere delle serie di if-then-else in maniera molto concisa.

La sintassi di case e' la seguente:

case *expression* **in**
pattern1)
statements;;
pattern2)
statements;;
...
esac

Ognuno dei pattern puo' essere costituito da pattern multipli separati dal carattere pipe (|). Se *expression* fa il match con uno dei pattern, allora gli *statement* corrispondenti vengono eseguiti. Se ci sono piu' pattern separati da pipe, allora l'espressione puo' fare il match con ognuno di essi. I pattern vengono controllati in ordine finche' qualcuno fa il match, se nessuno lo fa non accade nulla.

Esiste peraltro la possibilita' di specificare come ultimo pattern della serie il carattere *, in questo modo viene assegnata un'opzione di default se nessuna delle precedenti fa il match.

2.5.4 select

Select e' un costrutto disponibile sono nella bash[3] e nella Korn shell, e non ha un corrispondente in altri linguaggi.

[3] Versioni successive alla 1.14.3

Select permette di generare menu' molto semplicemente. La sintassi e' concisa, ma molto potente:

select *name [***in** *list]*
do
statements that can use $name
done

Come per il for se la lista viene omessa, il default e' "$@".
Select compie le seguenti azioni:

- Genera un menu' per ogni oggetto della *list*, formattato un un numero per ogni scelta
- Fa apparire un prompt dove deve essere inserito un numero
- Salva la scelta fatta nella variabile *name* e il numero selezionato nella variabile built-in REPLY
- Esegue gli statement del body
- Ripete il processo all'infinito

La variabile built-in *PS3* contiene la stringa del prompt che select utilizza; il suo valore di default e' "#?". E' possibile uscire da un loop select specificando il comando *break*. Break e' un costrutto che puo' essere utilizzato per uscire da qualsiasi struttura di loop, anche se deve essere utilizzato con parsimonia. Da un loop select e' possibile uscire anche con la combinazione di tasti CTRL-D.

2.5.5 While and until

Sono due costrutti simili, entrambi permettono l'esecuzione di una sezione di codice ripetitivamente quando (*while*) e finche' (*until*) una certa condizione diventa vera.

Sono costrutti utili soprattutto in combinazione con l'aritmentica degli interi.
La loro sintassi e' la seguente:

while *o* **until** *condition*
do
statements ...
done

La condizione e' in realta' una lista di statement come per if, l'exit status dell'ultimo viene preso come valore della condizione, anche in questo caso e' possibile l'utilizzo del costrutto test.

L'unica differenza fra while e until sta nel modo in cui la condizione viene gestita. Il loop while viene eseguito per tutto il tempo in cui la condizione e' vera. Until viene eseguito finche' la condizione e' falsa. La condizione di until, a differenza di altri linguaggi, viene controllata ad inizio ciclo e non alla fine.

2.6 Approfondimento sulle variabili

2.6.1 Typed variables

Finora abbiamo visto che alle variabili possono essere assegnati valori testuali. Le variabili possono pero' avere anche altri attributi.

E' possibile impostare gli attributi di una variabile con il built-in *declare* [4]. Un carattere - attiva un'opzione, mentre un + la disattiva. La seguente tabella riassume le opzioni disponibili:

Option	Meaning
-a	La variabile viene trattata come un array
-f	Mostra i nomi di funzione
-F	Mostra i nomi di funzione senza definizioni
-i	La variabile viene trattata come numero intero
-r	Trasforma la variabile in read-only
-x	Marca la variabile per l'export via environment

Impartire il comando declare senza argomenti sulla linea di comando, mostra tutti i valori delle variabili dell'environment. Un built-in che e' in stretta relazione con quelli visti e' *readonly*, che opera nello stesso modo come declare -r.

Le variabili dichiarate in una funzione sono locali ad essa, come se si fosse usato il costrutto *local*.

2.6.2 Integer Variables and Arithmetic

Un'espressione come *$((* and *))* viene interpretata dalla shell come unn espressione aritmetica e come tale il contenuto viene valutato. Le variabili nelle espressioni aritmetiche non necessitano di essere di essere precedute dal segno *$*, anche se non e' sbagliato farlo.

Le espressioni aritmetiche sono valutate all'interno delle *double quotes*, cosi' come le tilde substitution, le sostituzioni di variabili e le sostituzioni di comando.

Come regola generale, se si e' in dubbio e' meglio includere una stringa in single quotes, a meno che non contenga tilde o espressioni che coinvolgono un carattere $, nel qual caso e' meglio utilizzare le double quotes.

Gli operatori aritmetici sono gli stessi utilizzati dal C:

+ - * / % << >> & | ! ^

e nello stesso modo gli operatori relazionali

< > <= >= == != && ||

Le parentesi tonde possono essere utilizzate per raggruppare le sottoespressioni.

[4] Typeset e' un sinonimo, ormai obsoleto

2.6.3 Arithmetic variables and assignment

Come abbiamo visto e' possibile assegnare varibili aritmetiche per mezzo di declare. E' possibile valutare espressioni aritmetiche e assegnarle alle variabili mediante l'uso di *let*.

La sintassi e':

let *intvar=expression*

Non e' necessario utilizzare il costrutto $((and)) nell'espressione del comando let. Let non crea una variabile di tipo integer, ma semplicemente impone che l'espressione che segue l'assegnazione sia interpretata come se fosse di tipo aritmetico. Non deve esserci spazio attorno al segno =, ed e' buona norma circondare l'espressione con le quotes, in quanto i simboli matematici potrebbero essere interpretati erroneamente dalla shell.

2.6.4 Arrays

Un array e' simile ad una serie di caselle che contengono valori. Ogni casella e' detta *elemento* e ad ogni elemento si puo' accedere attraverso un indice numerico. Un elemento di un array puo' contenere una stringa o un numero e puo' essere utilizzato come ogni altra variabile. L'indice dell'array parte da 0. Cosi' per esempio il quinto elemento dell'array *prova* sara' referenziabile come *prova[4]*. Gli indici possono essere qualunque espressione aritmetica che restituisca un numero maggiore o uguale a zero. Un elemento di un array puo' essere assegnato in vari modi, per esempio:

prova[2]=test
prova[0]=test2
prova=([2]=test [0]=test2)
prova=(test2 [2]=test)

Un array viene creato automaticamente da ognuna di queste espressioni di assegnazione, per creare esplicitamente un array vuoto occorre usare declare -a.

Un elemento di un array viene referenziato con la sintassi:

${array[i]}

E' possibile anche utilizzare degli indici speciali: @ e *. Questi ritornano tutti i valori dell'array e funzionano come per i parametri posizionali. Quando il riferimento all'array e' in double quotes, utilizzando * si ottiene una sola parola che contiene tutti i valori dell'array separati dal primo carattere della variabile IFS, mentre @ espande i valori dell'array in parole separate.

I valori non assegnati degli array non esistono e se vengono referenziati corrispondono alla stringa null. E' possibile eliminare quasiasi elemento di un array con il costrutto unset.

3 The X Window System

Lo *X Window System* comprende tutto quel software che permette di lavorare con l'interfaccia video e far si che la parte grafica possa apparire sul monitor. Ma X va ben oltre tutto questo, infatti fornisce un interfaccia di flessibilita' quasi illimitata che permette ai programmi di mostrare della grafica, interagire

con l'utente, e scambiare dati con altri programmi grafici. Sia GNOME che KDE sono dei set di librerie e programmi che girano al di sopra di X.

3.1 La storia di X

X e' un sistema di interfaccia grafica a finestre completo che puo' girare, praticamente, su ogni tipo di computer, ma che ha avuto il proprio successo principalmente su UNIX. X offre un gran numero di opzioni sia per il programmatore che per l'utente.

X fu sviluppato in origine dal Project Athena presso il MIT, dal MIT stesso, IBM e Digital DEC. L'attuale vesione di X e' la 11 revision 6 (*X11R6*), che e' stata inizialmente rilasciata nell aprile del 1994 e successivamente aggiornata. Dal rilascio della Versione 11, X e' diventato lo standard de facto degli ambienti grafici UNIX.

Nonostante ne venga fatto un uso commerciale, l'X windows system resta distribuito sotto una licenza open source dall'Open Group. Una completa implementazione di X e' liberamente disponibile per i sistemi Linux. *X.org*[5], la versione che deriva direttamente dai sorgenti di X e che e' la piu' utilizzata nei sistemi Linux, supporta molteplici architetture e una notevole quantita' di sistemi operativi e schede grafiche. Oltre a queste versioni libere, esistono anche delle versioni commerciali di X per Linux. Vale la pena di notare come su una macchina server molto spesso X non venga neppure installato, visto che il controllo del sistema viene fatto per lo piu' remotamente.

3.2 I concetti di X

X e' basato sul modello client/server, nel quale l'X server e' un programma che gira sul sistema e gestisce tutti gli accessi all'hardware grafico. Un X client e' un applicazione che comunica con il server, inviandogli una serie di richieste. L'X server si prende cura di servire queste richieste.

E' importante notare che X e' un programma orientato alla rete, cioe' un X client puo' girare sia localmente (ovvero nella stessa macchina su cui gira il server) sia remotamente (su un qualunque sistema di una rete TCP/IP). L'X server resta in ascolto sulle socket locali e remote in attesa delle richieste dei client.

Altre caratteristiche di X sono la sicurezza (l'utente puo' deciderne il livello), la separazione modulare delle funzioni e il supporto per molte architteture.

L'X Windows System fa un netta distinzione fra il comportamento di un applicazione e la gestione della finestra. I client che girano sotto X vengono mostrati in una o piu' finestre sullo schermo. Comunque, come le finestre sono manipolate (poste sullo schermo, risimensionate, ecc.) e la loro decorazione (l'apparenza degli window frame) non vengono controllati dall'X server. Al contrario queste cose sono gestite da un particolare X client detto *window manager* che gira parallelamente a tutti gli altri X client. Lo window manager che un utente scegliera', controllera' in buona parte il "look&feel" di X. Tutti gli

[5] X.org e' una versione abbastanza nuova di X. A causa di una serie di contrasti nella comunita' di X Windows System, c'e' stata una biforcazione dello sviluppo. La stragrande maggioranza degli sviluppatori e' passata dalla dalla versione che in passato prevaleva XFree86 alla nuova X.org.

window manager sono altamente configurabili, i piu' moderni offrono una GUI per la personalizzazione.

Lo window manager non influenza cosa l'applicazione fa all'interno della finestra, ad esso viene solo affidato il compito di disegnare la decorazione, ovvero il contorno e i bottoni che permettono di chiudere, spostare e ridimensionare le finistre.

Ci puo' essere solo uno window manager per ogni X server. In teoria sarebbe possibile fare completamente a meno degli window manager, ma a questo punto non sarebbe possibile muovere le finestre sullo schermo e molte altre tipiche funzionalita', a meno che non fosse l'applicazione stessa a fornirle.

Oltre agli window manager ci sono i desktop environment, come GNOME e KDE. Essi sono una collezione di applicazioni e tool con un look&feel comune, oltre a molte altre proprieta'. In ogni caso anche i desktop environment su X hanno bisogno di uno window manager, in genere questi forniscono il proprio, ma non e' sempre cosi'.

3.3 Uno sguardo all'hardware

Attualmente ci sono due versioni supportate di X.org. La release piu' nuova e' la 7 (X11R7.1), ma viene ancora supportata anche la 6.9 (e in parte la 6.8), che ancora molte distribuzioni enterprise forniscono per motivi di stabilita'. Sul sito di X.org e' possibile reperire la lista dei chip grafici supportati e la documentazione specifica per ognuno di essi. Se dobbiamo testare quale chipset video abbiamo, il metodo migliore e' lanciare:

Xorg -configure

a questo punto X.org carichera' tutti i driver per i display grafici e alla fine del test scrivera' un file di configurazione iniziale che l'utente potra' modificare in base alla proprie esigenze. Va fatto notare come il progetto X.org abbia istituito una architettura dei driver completamente diversa dalla precedente, molto piu' flessibile. Le interfacce grafiche che utilizzano un chipset riconosciuto sono normalmente supportate su tutti i tipi di bus (AGP, PCI, PCI-X, PCI Express). Tutti i chipset sono supportati con 256 colori, alcuni sono supportati in mono e a 16 colori, molti altri con profondita' di colore superiori.

Oltre ai vari chipset c'e' anche supporto per il framebuffer devive, che viene fornito dal kernel (a partire dalla versione 2.2) per mezzo del driver *fbdev*. Questo driver viene utilizzato qualora il chipset non sia supportato dall'X server, sebbene le performance non siano ottimali. Su qualche hardware anche il framebuffer permette di avere grafica accelerata.

Un problema che gli sviluppatori di X.org hanno dovuto franteggiare e' che alcuni costruttori di interfacce video utilizzano dei metodi non standard per determinare le frequenze di clock necessarie a pilotare l'interfaccia. Altri, invece, non rilasciano le specifiche su come programmare l'hardware oppure richiedono la sottoscrizione di un non-disclosure agreement per poter accedere a queste informazioni. Questo naturalmente restringerebbe la libera distribuzione di X.org e quindi il team di sviluppo non lo ha mai accettato.

3.4 Installazione di X.org

X.org non fornisce nessuna distribuzione binaria, ma quelle che vengono fornite con le distribuzioni sono ottime. Sul sito ftp di X.org e' possibile trovare il

codice sorgente e le istruzioni su come crearsi una distribuzione binaria.

Scrivere un file di configurazione (che puo' chiamarsi *xorg.conf* oppure *XF86Config-4* a seconda delle versioni e delle distribuzioni) da zero puo' essere piuttosto complesso e non e' raccomandato. Esistono almeno tre modi per creare un file di configurazione:

- Il primo e' di usare il set up tool che la distribuzione fornisce. Se esso non fosse sufficiente, in genere viene fornito un programma chiamato *xorgcfg*. Si tratta di un programma per l'installazione grafica che funziona anche sul terminale, visto che ancora X non e' stato configurato.
- Come secondo passo, si puo' cercare di utilizzare il comando *Xorg -configure*, che avvia il server in una modalita' in cui cerca di reperire il massimo delle informazioni sull'hardware e poi scrive un file di configurazione di base, che dovrebbe essere in grado di far partire l'X server, sebbene possa essere ulteriormente affinato.
- Se ogni altro metodo fallisce e' sempre possibile tentare con un altro tool testuale, che si chiama *xorgconfig* e che dovrebbe essere installato insieme a X.org. Esso guida l'utente attraverso una serie di passaggi e domande relative all'hardware, alla fine produce un file di configurazione base.

3.5 Configurare X.org

Configurare X.org e' un attivita' che puo' durare molto tempo se si cercano di ottimizzare al massimo le prestazioni.

Il file di configurazione *xorg.conf* in genere si trova nella directory */etc/X11/*. Partendo da un file base creato con uno dei metodi appena visti, la prima cosa da fare e' quella di avviare Xorg a bassa risoluzione (per es. 640x480), che dovrebbe funzionare su tutte le interfacce video, se tutto funziona e' possibile poi espandere la configurazione. E' comunque sempre consigliabile leggere la documentazione per lo specifico chipset che stiamo utilizzando.

Il principale file di configurazione di Xorg si chiama /etc/X11/xorg.conf, esso contiene informazioni sul mouse, i parametri dell'interfaccia video e cosi' via. Questo e' il file da modificare per adattarlo alla configurazione locale.

La man page di xorg.conf descrive nei dettagli i parametri del file.

Sebbene i parametri di un file di configurazione possano differire molto da sistema a sistema e anche il formato stesso del file vari fra le versioni di Xorg, possiamo dare un'occhiata generale alla sintassi.

Ogni sezione del file Xorg.conf viene circondata da una coppia si linee:

Section "section-name"
...
EndSection

All'interno di ogni sezione ci possono essere molte linee.

Una sezione importante e' quella *Files* che contiene, in genere molte linee.

Ogni linea *FontPath* configura il path di una directory che contiene i font X11. Se viene aggiunta la stringa *:unscaled* alla fine della linea, allora i font relativi non saranno riscalati, questo perche' i font molto riscalati hanno un pessimo aspetto.

Altre linee possono contenere un *RgbPath*, ovvero il database dei colori RGB. Altre ancore un *ModulePath*, che punti ad una directory con dei moduli caricabili dinamicamente.

Una successiva sezione che e' possibile trovare e' *ServerFlags*, che in genere ha pochi elementi, in cui e' possibile specificare cose come: fai partire il server anche se il mouse non e' presente. In ogni caso la maggior parte delle opzioni dovrebbero essere rilevate automaticamente all'avvio.

La sezione successiva e' *Module*, nella quale e' possibile caricare dinamicamente dei moduli aggiuntivi, per es. per il supporto di hardware particolare o speciali librerie grafiche. Questa sezione viene utilizzata anche per caricare la libreria di supporto freetype e i moduli per il 3D.

Le sezioni successive sono *InputDevice*. In genere ce ne sono almeno due, una per la tastiera e una per il mouse, piu' altre se ci sono ulteriori dispositivi.

Qui e' possibile stabilire il layout della tastiera (*kbd)* e altre caratteristiche.

La sezione *mouse* dice al server come e' collegato il mouse, che genere di mouse e' (*Protocol*) e altri dettagli operativi. Esiste la parola magica *Auto* che fara' in modo che sia il server a cercare di autoconfigurare il mouse allo startup.

La sezione successiva e' *Device*, che specifica i parametri dell'interfaccia video. Se una macchina ha piu' di una scheda video, ci saranno molteplici sezioni di questo tipo.

La entry *BoardName* e' semplicemente un nome simbolico. Allo stesso modo *VendorName*. Anche *Identifier* e' una stringa che puo' essere scelta liberamente ma che deve corrispondere alla stringa *Device* utilizzata nelle sezioni successive del file di configurazione. Solitamente si scelgono i nomi *Device [0], Device[1]* ...

BusID identifica l'interfaccia video in termini di hardware sul bus PCI, l'espressione e' del tipo *PCI:1:0:0* o la piu' concisa *1:0:0*. Per avere questa informazione si puo' utilizzare il comando *Xorg - scanpci*.

Segue una voce *Screen* che e' obbligatoria per le interfacce multi-head, nelle normali interfacce il valore 0 e' adeguato.

Driver e' sicuramente la voce piu' importante perche' determina il driver che deve essere caricato dall'X server. Un buon sistema per scoprire il nome del driver e' quello di usare uno dei programmi di configurazione, oppure lanciare l'X server nel seguente modo: *X -probeonly*

Questo mostrera' tutte le informazioni che l'X server ha raccolto sull'hardware e fra le altre anche il driver video che pensa di utilizzare. Ci sono moltissime voci che e' possibile configurare in questa sezione, come il chipset, il RAMDAC e altre proprieta' hardware, ma l'X server in genere e' in grado di trovarle da solo.

La sezione che segue e' *Monitor*, che specifica le caratteristiche del monitor, come per ogni altra sezione, ce ne puo' essere piu' di una.

La linea *Identifier* e' un nome arbitrario per la sezione. *HorizSync* specifica le frequenze orizzontali di sincronizzazione valide per il monitor in kHz. Se il monitor e' di tipo multisync, questo e' in genere un range di valori. Questi parametri vengono presi o da un database fornito dalla distribuzione oppure dal manuale del monitor.

VertRefresh specifica il rate di refresh verticale valido per il monitor in Hz. Anch'esso puo' essere espresso come range.

I valori di *HorizSync* e *VertRefresh* vengono usati per fare un doppio controllo e vedere che le frequenze di pilotaggio del monitor non eccedano i limiti posti,

prevenendo danni all'hardware.

E' possibile anche utilizzare delle voci *Mode* e direttive *Modeline* nella sezione *Monitor* per specificare le risoluzioni del monitor. Al momento queste voci non sono piu' necessarie ed sono state trasferite nella successiva sezione *Modes*.

La sezione *Modes*, di cui ce ne dovrebbe essere una per ogni monitor configurato, e' la lista dei vari video modes che l'X server dovrebbe supportare. La linea *Identifier* si riferisce al nome specificato nella sezione *Monitor*. Le linee *Modeline* specificano un modo video ognuna. Il formato e':

Modeline name dot-clock horiz-values vert-values

dove *name* e' una stringa arbitraria, *dot-clock* e' la frequenza di clock di pilotaggio o il dot clock associato con la risoluzione, il dot clock e' in genere specificato in MHz ed e' il rate a cui la scheda video deve inviare i pixel al monitor a questa risoluzione. *Horiz-values* e *vert-values* sono quattro valori ognuno, essi specificano i parametri di pilotaggio del cannone elettronico del monitor.

Determinare i parametri *Modeline*, se questo non viene fatto automaticamente da programma di configurazione, non e' banale. In caso di necessita' puo' essere utilizzato il tool *xvidtune*.

X.org ha built-in la cosiddetta modalita' VESA di timing del monitor, cosi' da poter lavorare anche con la sezione *Modes* assente. I timing VESA sono valori standard per *Modeline* che dovrebbero funzionare sulla maggior parte dei display, sebbene senza nessuna ottimizzazione.

Vale la pena di notare come il parametro *name* della linea *Modeline* debba poi essere riutilizzato successivamente, ma puo' essere una stringa qualunque.

Per ogni linea *Modeline* il server controllera' che il modo ricada all'interno dei valori specificati con *HorizSync* e *VertRefresh*. Altrimenti il server si lamentera' all'atto dell'avvio.

La successiva sezione *Screen* specifica la combinazione monitor/interfaccia video. Questa sezione mette insieme le definizioni device, screen e monitor ed elenca le profondita' di colore da utilizzare con i video modes.

Infine la sezione *ServerLayout* mette assieme le cose identificando una configurazione che consiste di una o piu' sezioni *Screen* e una o piu' sezioni *InputDevice*.

Nella configurazione possono esistere altre sezioni, ma esse sono opzionali e per esse si rimanda alla documentazione.

3.5.1 Lanciare X

Per lanciare X, se /usr/X11R6/bin e' nel nostro PATH, e' sufficiente:

startx

esso non e' altro che un frontend a *xinit*. Quest'ultimo puo' essere utilizzato, ma costringe ad avviare tutti i programmi necessari manualmente.

Il comando fa partire l'X server e lancia i comandi che trova nel file *.xinitrc* nella home directory. *.xinitrc* e' semplicemente uno shell script che contiene gli X client da lanciare. Se il file non esiste, viene utilizzato il default di sistema /usr/X11R6/lib/X11/xinit/xinitrc. E' possibile cambiare il display all'avvio di X fornendo un diverso .xinitrc nella home directory.

3.5.2 Terminare X

E' possibile uscire da X semplicemente chiudendo tutti client che sono connessi al server e poi chiudendo anche il processo X. A volte ci si puo' trovare in situazioni di emergenza in cui sia necessario chiudere una sessione X bloccata, per fare questo esiste una combinazione di tasti di kill: CTRL-ALT-BACKSPACE. Naturalmente e' possibile configurare l'X server per ignorare questa combinazione di tasti, ma in genere non viene fatto, perche' puo' salvare da molte situazioni complicate. Va fatto notare che premere CTRL-ALT-BACKSPACE comporta in sostanza un crash del server X e quindi e' una possibilita' da utilizzare solo quando davvero necessario.

Oltre a questo e' bene comprendere che il meccanismo dell'interfaccia grafica completamente in user space di UNIX (cioe' un normale processo) sia particolrarmente robusta rispetto a soluzioni in kernel space di altri sistemi operativi. Sebbene quest'ultimo approccio possa comportare dei vantaggi dal punto di vista delle prestazioni (vengono eliminate tutte le system call ed i conseguenti context switch), espone al rischio che un bug dell'interfaccia grafica possa bloccare l'intero sistema.

3.6 XDM e XDMCP

Un metodo alternativo per avviare l'X server, oltre all'utilizzo di xinit e' l'uso dell'X Display Manager. Se quello che vogliamo e' avere X sempre attivo sul display, l'amministratore puo' provvedere al setup di XDM. Questo e' un programma in genere avviato al boot (dal processo init, per certi runlevel) e che si occupa di mantenere il server in esecuzione e gestire il login degli utenti.

Se XDM e' attivo, all'avvio apparira' una schermata grafica in cui sara' possibile effettuare il normale processo di login. Appena il login sia stato completato con successo, xdm si premurera' di avviare l'ambiente X di ogni utente. Se un utente ha un file *.xsession* nella propria home directory, xdm lo trattera' come uno shell script da eseguire per far partire i client X iniziali, lo window manager e le altre caratteristiche dell'ambiente X.

Il display manager e' in grado di fare anche piu' di questo, esso permette di accedere ad una sessione grafica da remoto. Il servizio offerto in questo caso si chiama XDMCP e si pone in ascolto per default sulla porta 177.

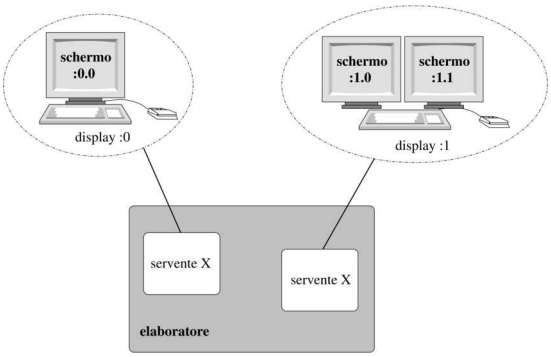
Il calcolatore remoto in questo caso deve avviare il proprio server X, e connettersi poi al calcolatore che offre il servizio XDMCP e da questo momento lavorare sulla sessione X remota.

Il servizio xdm viene configurato e avviato per mezzo dei file contenuti nella directory /etc/X11/xdm. Di particolare interesse e' il file xdm-config, che fra le altre cose permette anche l'avvio di XDMCP, che, in genere, e' disabilitato per default.

I desktop environment Gnome e KDE forniscono i propri display manager, che nella maggior parte delle distribuzioni hanno sostituito il classico xdm. Quello di Gnome si chiama GDM e viene configurato con i file contenuti nella directory /etc/X11/gdm/. Quello di KDE si chiama KDM e viene configurato con i file contenuti in /etc/kde/kdm/. In molte distro il display manager utilizzato dipende da quello che e' il default environment del sistema, ma naturalmente questa impostazione puo' essere modificata facilmente.

3.7 Un breve approfondimento

Dal punto di vista di X, l'hardware è ciò che consente di interagire con questo sistema grafico (nel senso che il resto non è di sua competenza). Si tratta della tastiera, dello schermo grafico e del dispositivo di puntamento. In pratica il ruolo di X è quello di controllare tutto questo.



All'interno di un elaboratore possono funzionare teoricamente più server grafici per controllare altrettante stazioni grafiche di lavoro. Inoltre, sempre teoricamente, una stazione grafica può utilizzare più di uno schermo grafico contemporaneamente.

Nel gergo di X la stazione grafica è il display, che viene identificato da un numero a partire da zero, nella forma *:n*. Se una stazione grafica è dotata di più di uno schermo, quando si deve fare riferimento a uno di questi occorre aggiungere all'indicazione del numero della stazione grafica quello dello schermo. Anche in questo caso, il primo corrisponde a zero. La forma diventa quindi *:n.m*, dove n è la stazione grafica e m è lo schermo. La figura dovrebbe chiarire il meccanismo. Il valore predefinito di stazione grafica e schermo è zero, per cui, quando non si specificano queste informazioni, si intende implicitamente lo schermo :0.0.

X è trasparente nei confronti della rete. Un programma client può utilizzare i servizi di un server remoto, interagendo con la stazione grafica di quel server. Questo tipo di utilizzo richiede comunque una forma di autorizzazione o autenticazione, per motivi di sicurezza.

Quando si vuole identificare uno schermo particolare di un certo elaboratore nella rete, si antepone alle coordinate il nome o l'indirizzo di quell'elaboratore: *nodo:n.m*.

Un programma client può connettersi con un server X, sia locale, sia remoto. Per una connessione remota occorre stabilire un collegamento. Il server X resta normalmente in ascolto sulla porta 6 000 + n, dove n rappresenta il numero della stazione grafica, ovvero del server X.

X11R6 presenta una serie di meccanismi di sicurezza per l'accesso dei client al display, questi sono:

- *Host* Access Simple host-based access control.
- *MIT-MAGIC-COOKIE-1* Shared plain-text "cookies".
- *XDM-AUTHORIZATION-1* Secure DES based private-keys.

- *SUN-DES-1* Based on Sun's secure rpc system.
- *MIT-KERBEROS-5* Kerberos Version 5 user-to-user.

Xdm inizializza l'access control per il server e mette le informazioni di autorizzazione in un file accessibile all'utente. In genere la lista di host le cui connessioni sono sempre accettate dovrebbe essere vuota, cosi' che solo i client espressamente autorizzati possano connettersi. Quando viene aggiunto un client alla host list (con il comando *xhost*), il server non esegue piu' nessuna operazione di verifica delle autorizzazioni per esso.

Il file da cui vengono estratte le informazioni di autorizzazione (puo' essere modificato) e' in genere *.Xauthority*. Esso viene utilizzato e aggiornato da Xdm. La gestione dei file di autorizzazione avviene per mezzo di *xauth*.

In genere i metodi piu' utilizzati per l'autenticazione sono i primi due:

- Host Access: ogni client nella host access control list puo' connettersi all'X server. E' un sistema utilizzabile in situazioni di totale fiducia fra i soggetti. La lista degli host viene salvata all'interno del server X e puo' essere cambiata con il comando *xhost*. Quando si utilizzano livelli di sicurezza superiori, questa lista dovrebbe essere in genere vuota.
- MIT-MAGIC-COOKIE-1: in questo caso il client invia un "cookie" di 128 bit assieme alle informazioni per il setup della connessione. Se il cookie corrisponde a uno di quelli in possesso del server, allora la connessione viene accettata. La copia utente del cookie viene salvata nel file *.Xauthoriry* nella home directory dell'utente. Xdm passa automaticamente il cookie al server per ogni login session e lo salva nello user file al login.

Per ulteriori informazioni e per un approfondimento sugli altri metori di autenticazione si rimanda a Xsecurity(7).

4 I Desktop Manager

La maggior parte delle distribuzioni Linux prevedono la presenza di un ambiente grafico completo noto come *environment* o *Desktop Manager*.

I due principali desktop manager disponibili sono KDE (K Desktop Environment) e GNOME.

Naturalmente per un server non avrebbe senso installare un intero sistema grafico integrato e non ha significato in un'altra serie di situazioni. Ma per il comune lavoro su una workstation utilizzare uno dei display manager puo' essere molto appagante. Naturalmente a spese delle risorse di sistema che vengono richieste.

Un utente moderno di un sistema operativo chiede in genere:

- La possibilita' di associare le applicazioni alle icone e di lanciarle automaticamente senza doverle speficicare
- Il copia e incolla di testo e immagini da una finestra all'altra
- La possibilita' di salvare e rispristinare le sessioni, in modo da poter ricominciare il lavoro da dove si era lasciato

- Un meccanismo di help del sistema, come thumbnails e tooltips
- La disponibilita' di background, screen saver e temi
- Permettere una ampia personalizzazione dell'ambiente, ma offire un default con cui gli utenti si sentano comunque a proprio agio

Poiche' Gnome e KDE sono stati progettati per essere intuitivi, prendendo in prestito molti concetti da altri ambienti grafici, il loro utilizzo dovrebbe essere semplice per la maggior parte degli utenti.

4.1 KDE

KDE e' un progetto opensource iniziato nel 1996, basato sul toolkit Qt della Trolltech e conseguentemente sul linguaggio C++.

KDE utilizza una tecnologia a componenti della KParts che permette di includere (embed) un'applicazione in un'altra trasparentemente.

Lo sviluppo di KDE e' molto attivo, il team rilascia con cadenza di alcuni mesi una official release, considerata stabile. Essa viene fornita come sorgenti e sono le distribuzioni che costruiscono i pacchetti binari relativi.

Al momento la serie sviluppata e' KDE versione 3.

4.1.1 Concetti generali

Uno degli obiettivi di KDE era quello di rendere tutto l'ambiente configurabile per mezzo della GUI. Al di sotto di questo livello ci sono una serie di file di testo, nel semplice formato *parameter=value*.

Le carattestiche che offre KDE sono numerosissime, vale la pena di notare il drag&drop particolarmente efficiente, sia nei confronti di risorse locali che remote.

Sebbene KDE abbia a disposizione le man page, queste sono orientate ad un pubblico di programmatori. La documentazione e' quindi rilasciata in formato HTML (generata da file XML) e ha un viewer dedicato (KDE Help Center).

KDE ha integrato nell'ambiente un session manager, esso permette all'uscita dell'ambiente il salvataggio delle sessione per tutte le applicazioni che gestiscono il session management. KDE fa un uso estensivo di questo componente.

KDE supporta l'anti-alias che viene fornito come caratteristica dell'X server.

KDE contiene un window manager, *kwin,* questa naturalmente e' solo una parte di KDE, oltre a questo ci sono il file manager, web browser, panel, pager e control center per la configurazione del desktop e molto altro. E' possibile anche eseguire KDE con un altro window manager.

KDE viene fornito con una grandissima quantita' di applicazione integrate nell'ambiente. Tutte le applicazioni X funzioneranno in KDE, anche se per alcune non saranno disponibili tutte le funzioni delle applicazioni integrate. E' possibile istruire KDE ad integrare al massimo le applicazioni X esterne all'interno dell'ambiente.

4.1.2 Installazione

Abbiamo visto che e' possibile compilare KDE dai sorgenti, ma la maggior parte delle distribuzioni lo forniscono in forma compilata. KDE e' in genere costituito da una serie di pacchetti, che rispecchiano la modularizzazione dei sorgenti:

- *aRts*: e' il diminuitivo di "a real-time sequencer" ed e' il fondamento della maggior parte delle capacita' multimediali di KDE.
- *kdelibs*: The KDE libraries. Contengono il basic application frame, un certo numero di GUI widgets, il sistema di configurazione, il sistema di HTML display e altro. Senza questo pacchetto KDE non puo' lavorare.
- *kdebase*: In questo pacchetto ci sono le applicazioni base di KDE che costituiscono il KDE desktop, incluso il file manager/web browser, window manager e panel.
- *kdegames*: Un certo numero di giochi.
- *kdegraphics*: Un gruppo di applicazioni orientate alla grafica, come: viewer, PostScript viewer, icon editor.
- *kdeutils*: Un gruppo di tool per il comune lavoro, come: text editors, calculator, printer managers, ecc.
- *kdemultimedia*: Questo pacchetto contiene programmi multimediali, come: CD player, MIDI player, MP3 player, ecc.
- *kdenetwork*: In questo pacchetto sono contenuti programmi per l'uso su Internet, come: news reader, e i network management tools.
- *kdeadmin*: Questo pacchetto contiene programmi per l'amminstratore di sistema, come: user manager, run-level editor, backup program.
- *kdepim*: Al giorno d'oggi e' considerato il fulcro di KDE, kdepim contiene software per il personal information management, in particolare: Kontact l'utilita' di pianificazione, KOrganizer, il KDE email package KMail, l'address book, il software di sincronizzazione per il PDA, ecc.
- *kdeedu*: Come si puo' evincere dal nome, il pacchetto contiene una serie di software educativi.
- *kaccessibility*: Questo pacchetto coniene una serie di tool che rendono possibile, o piu' semplice, l'uso del computer alle persone con disabilita'. L'obiettivo di KDE e' di essere totalmente compatibile con l'*Americans with Disabilities Act*.
- *kartwork*: Questo pacchetto include artwork per kde, fra cui: sets di icone, wallpaper, ecc.
- *kde-i18n*: Ci sono una serie di pacchetti che inizano con KDE-i18n-. Essi contengono le traduzioni per una serie di linguaggi. Il default e' American English.
- *kdetoys*: Questo pacchetto contiene una serie di piccoli programmi che non hanno un preciso scopo, ma che sono divertenti o piacevoli da vedere. Per esempio: AMOR, Amusing Misuse of Resources.
- *kdewebdev*: E' un pacchetto dedicato agli sviluppatori di pagine web. Contiene oggetti come: Quanta HTML editor.

- *koffice*: KOffice non e' altro che una office suite.
- *Developer tools*: Ci sono una quantita' di pacchetti per gli sviluppatori di applicazioni KDE. KDEsdk contiene tools, scripts, e information per gli sviluppatori KDE. KDEbindings contiene collegamenti per creare programmi KDE in linguaggi diversi da C++ e infine, kdevelop e' un ambiente completo di sviluppo non solo per apllicazioni KDE.

Oltre a queste applicazioni fornite dal team ufficiale di KDE, ce ne sono molte altre disponibili in rete.

4.1.3 KDE Panel e KDE Menu

Quando KDE viene avviato, lungo il bordo inferiore dello schermo viene mostrato il cosidetto *panel*. Esso fornisce molte funzioni. Al primo avvio KDE propone (ma dipende dalla distribuzione) un pannello che permette di configurare molti aspetti.

KDE fornisce un certo numero di *workspace* che sono accessibili dai bottoni al centro del panel.

Ci sono molte cose che e' possibile fare con le finestre di KDE, ma concentriamo l'attenzione al menu che si apre premendo il bottone (*button*) piu' a sinistra del pannello. Questo e' quello che si chiama *K menu*. Assieme alle voci per configurare il panel e K menu, troviamo tutte le applicazioni KDE installate, raggruppate in sottomenu'.

Se ci fosse la necessita' di integrare ulteriori applicazioni del sistema in KDE, e' possibile utilizzare *KAppfinder*, che cerchera' le applicazioni presenti nel sistema, che hanno una corrispondenza nel suo database e le integrera' con il desktop. Per fare questo generera' un cosiddetto *.desktop* file per l'applicazione. E' sempre possibile scrivere uno di questi file a mano o seguire una procedura guidata se KAppfinder non avesse una corrispondenza nel database.

Al panel e' possibile aggiungere e togliere i quick launcher per ogni applicazione presente nel Kmenu; oltre alle applicazioni ci sono altri oggetti, le *applets*, piccoli programmi progettati per lavorare all'interno del panel e che non possono funzionare da sole.

4.1.4 Il KDE Control Center

La configurazione di KDE avviene per mezzo del *KDE Control Center* che puo' essere avviato da K menu. La configurazione avviene mediante l'accesso ad una serie di sezioni, come e' possibile vedere nella seguente figura:

4.2 GNOME

Cosi' come KDE, anche Gnome e' un completo ambiente integrato, dal desktop ad una serie di applicazioni. Anche Gnome puo' eseguire ogni tipo di applicazione X e cosi' come KDE si basa sugli standard di *freedesktop.org*. In questo senso e' possibile eseguire senza grossi problemi le applicazioni di un ambiente nell'altro.

Il primo obiettivo di Gnome e' sempre stato quello della facilita' d'uso, quindi tutte le applicazioni devono aderire alle human interface guidelines del team di sviluppo. Lo sviluppo delle applicazioni Gnome viene fatto in molti linguaggi e le Gnome libreries sono utilizzate anche esternamente al progetto. La base da cui e' partito il progetto sono state le librerie GTK di Gimp.

Il Desktop di Gnome e' stato progettato per essere familiare con chiunque abbia lavorato con un computer. Sebbene sia possibile modificare le impostazioni in molti modi, il desktop di default avra' i pannelli in alto e in basso. I pannelli sono fra gli elementi piu' importanti di Gnome perche' sono molto versatili e permettono una vasta gamma di interazioni con il sistema.

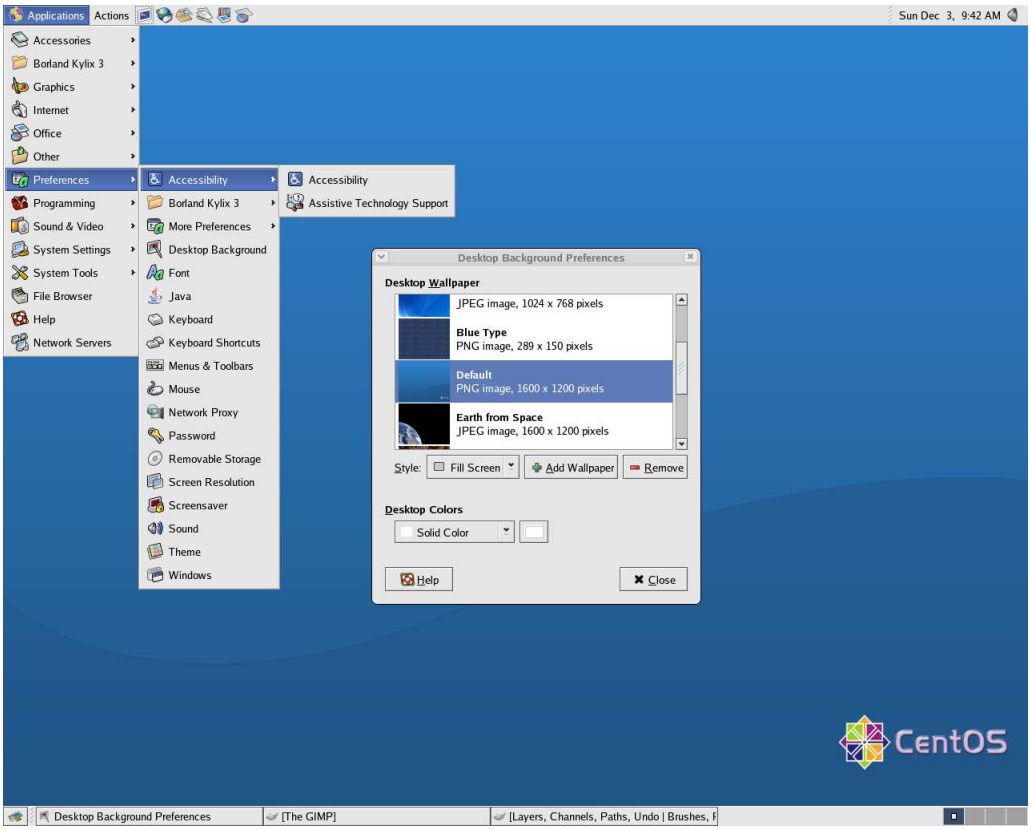
4.2.1 La configurazione di GNOME

GConf e' un sistema di configurazione, basato su XML centralizzato per le applicazioni desktop. Esso permette alle applicazioni di condividere tutta una serie di preferenze e utilizza un demone per notificare alle applicazioni quando le

preferenze cambiano, cosi' non e' necessario riavviare l'applicazione per vedere gli effetti.

GConf puo' essere utilizzato per il meccanismo di desktop lock down con un livello di granularita' nettamente superiore a quello di X. Un amministratore puo' bloccare le impostazioni di GConf, in modo da permetterne solo alcune.

Per operare sulla configurazione e' possibile editare manualmente il file *~/.gconf*, ma esiste un tool dedicato gconf-editor. Esistono peraltro una serie di tool integrati, disponibili generalmente nel menu preferences, che permettono di personalizzare vari aspetti del desktop environment.



4.2.2 Il pannello di GNOME

La configurazione che viene fornita per default, presenta un pannello sottile lungo il lato superiore ed inferiore dello schermo. Il pannello in alto presenta un set di menu' a sinistra oltre a vari bottoni (quick launchers) e all'orologio. Il pannello inferiore contiene l'applet con la lista delle finestre, in modo da poter passare facilmente da una all'altra.

E' possibile creare nuovi pannelli e modificare le impostazioni di quelli esistenti.

References

[1] Appunti di informatica libera - Appunti Linux Copyright © 1997-2000 Daniele Giacomini Appunti di informatica liberaCopyright © 2000-2006 Daniele Giacomini Via Morganella Est, 21 – I-31050 Ponzano Veneto http://na.mirror.garr.it/mirrors/appuntilinux/HTML/a2.htm

[2] By Cameron Newham - Learning the bash Shell, Third Edition (O'Reilly)

[3] By Matthias Kalle Dalheimer, Matt Welsh - Running Linux, Fifth Edition (O'Reilly)

[4] Advanced Bash-Scripting Guide - http://www.tldp.org/LDP/abs/html/index.html

[5] Bash Guide for Beginners - http://www.tldp.org/LDP/Bash-Beginners-Guide/html/index.html

Contents