Sistemi operativi

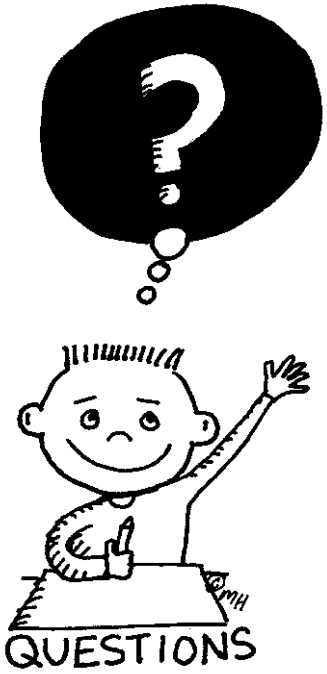
Corso di Laurea Triennale in Ingegneria Informatica

Lezione 9

- Shell scripting
- Gestione software in Debian
- Make

Fabio Del Vigna - fabio.delvigna@for.unipi.it

Domande sulle lezioni passate?



Sommario

- Come creare uno script Shell
- Variabili
- Parametri posizionali
- Operatori
- Strutture di controllo
- Opzioni

Introduzione

- Shell
  - Un interprete di comandi
  - Per esempio – sh, bash, csh, tcsh, kosh, zsh, ...
- Script Shell
  - Un insieme di comandi Unix salvati all'interno di un file di testo, dei quali si controlla il flusso di esecuzione
- La capacità di scrivere script dipende da
  - I comandi Unix conosciuti
  - Quanto “bene” si mettono insieme per svolgere il compito desiderato

Quando usare uno script Shell

- **NO**
  - Task ad alto carico computazionale
  - Portabilità tra piattaforme
  - Operazioni su file
  - Bisogno di porte o socket

- **SÌ**
  - Script di sistema (inizializzazione)
  - Automazione task di amministrazione
  - Prototipizzazione applicazioni

Creare uno script shell

Creare uno script shell

- Editare il file `hello.sh`

```
#!/bin/sh ...
echo "Hello World!"
...
```

- Rendere il file eseguibile
  - $ chmod +x hello.sh
- Eseguire hello.sh
  - $ ./hello.sh → esegue se il file è eseguibile
  - $ sh hello.sh → esegue il file usando l'interprete "sh"
- Debug hello.sh
  - $ sh -x hello.sh

Sharp bang e commenti

- Con quale interprete eseguire lo script
  - #! → Sharp-Bang (o ShaBang), è un numero magico a due byte
  - Seguito dall'indicazione con path completo dell'interprete da usare (es /bin/sh → #!/bin/sh)
  - Alternative – sh, bash, csh, tcsh, kosh, zsh, (perl), ...
- Commenti
  - # Questo è un commento

```
#!/bin/bash

y=0
for x in $@
do
```

Struttura di base di uno script

- Uno script è generalmente composto da
  - Indicazione interprete da usare
  - Righe di commento
  - Istruzioni da eseguire

```
#!/bin/sh

#Commento

<Istruzioni da eseguire>

...
```

Parole speciali

- L'interprete shell (Bash nel nostro caso) interpreta alcune sequenze di caratteri con un significato speciale
- Tali parole (quasi tutte comandi) sono:

```
! case do done elif else if fi for
function if in select then until while
{ } time [[ ]]
```

- Per usarle al di fuori del significato speciale bisogna usare accortezze, come apici o con caratteri di escape “\”

hello.sh

```
#!/bin/sh
#
# Hello world in a Bash script

EXIT_SUCCESS=0

echo -e "Hello world\n"

exit $EXIT_SUCCESS
```

Assegnamento a variabile

Uso di una variabile

Exit è un comando shell, che serve a comunicare con l'ambiente esterno

Variabili

- Assegnamento
  - VAR=0
- Lettura da tastiera
  - read VAR
- Nella dichiarazione della variabile si usa il nome privo di $, mentre va usato quando si riferisce la variabile
  - ${VAR} per riferirisi al valore

```
read VAR
echo "Il valore di VAR è ${VAR}"
```

Variabili d'ambiente 1/2

- Le **variabili d'ambiente** mantengono il loro valore nel tempo (esternamente all'esecuzione di uno script)
- L'esportazione delle variabili si ottiene con export, ma tale operazione ha effetto solo sui processi figli di chi invoca `export (setenv, printenv)`
- Le **variabili di shell** disponibili sono visibili con il comando `set (set)`

Variabili d'ambiente 2/2

- Alcune delle variabili disponibili sono
  - `$HOME – la home dir dell'utente`
  - `$HOSTNAME – hostname del calcolatore`
  - `$LOGNAME – username usato nel login`
  - `$IFS – lista separatori di campo`
  - `$PATH – la lista delle dir dei comandi`
  - `$PWD – dir di esecuzione dello script`
  - `$SHELL – la shell di default dell'utente`

```
echo -e “hostname:$HOSTNAME”
echo -e “logname:$LOGNAME”
```

Parametri 1/2

- Parametri posizionali
  - $0, $1, …, $n
    - ⇨ In $0 è posto il nome dello script
    - ⇨ Da $1 a $n sono i parametri passati allo script

```
#!/bin/sh
#parametri.sh
echo "uno:${1} due:$2 tre:$3"

$> ./parametri param1 param2
  param3
```

Parametri 2/2

- Altri parametri disponibili
  - `$$ - process id dello script`
  - `$? - valore di ritorno di un comando, funzione o dello stesso script`
  - `$* - stringa contenente tutti i parametri posizionali passati`
  - `$@ - insieme di tutti i parametri posizionali passati`

Operatori e condizioni

- Per la valutazione delle condizioni esistono in bash vari tipi di operatori
  - Operatori su numeri
    - ⇨ Operatori di confronto
    - ⇨ Operatori aritmetici
  - Operatori logici
  - Operatori su stringhe
  - Operatori su file
  - Operatori di Input / Output
  - Operatori su bit

Operatori su numeri - confronto

- Operatori di confronto
  - `-eq – “EQual”`
  - `-ne - “Not Equal”`
  - `-gt - “Greater Than”`
  - `-ge - “Greater or Equal”`
  - `-lt - “Less Than”`
  - `-le - “Less or Equal”`

```
es. [ “$NUM1” -eq “$NUM2” ]
es. [ 2 -lt 7 ] è vera
```

Operatori su numeri

- Operatori aritmetici
  - `+ - somma due numeri`
  - `- - sottrazione tra numeri`
  - `* - moltiplicazione tra numeri`
  - `/ - divisione intera tra numeri`
  - `** - esponenziazione (2**3 = 2^3)`
  - `% - resto intero divisione`

```
es. echo $(( 2 + 3 ))
```

Attenzione agli spazi!!!

Operatori logici

- Operatori logici disponibili
  - `! - negazione, NOT logico`
  - `-a – AND logico`
  - `-o – OR logico`

```
es. [ -e $FILE1 -a ! -e $FILE2 ]
```

Attenzione: non confonderli con gli operatori && e ||, usati nella concatenazione di comandi (per ottenere il "corto circuito")

Operatori su stringhe

- Operatori su stringhe
  - Unari – veri se …
    - `-n – lunghezza stringa maggiore zero`
    - `-z – lunghezza stringa zero`
  - Binari
    - `== o = - uguaglianza stringhe`
    - `!= - differenza stringhe`
    - `< - lessicograficamente minore`
    - `> - lessicograficamente maggiore`

`Attenzione nell'uso di < e >, racchiuderli tra virgolette!`

Operatori su file

- Alcuni operatori su file
  - `-e – vero se file esiste`
  - `-b – vero se file è block device`
  - `-f – vero se file è regolare`
  - `-r – leggibilità file`
  - `-w – scrivibilità file`
  - `-x – eseguibilità file`
  - `-s – dimensione file non zero`
  - `-L – argomento è link simbolico`
  - `file1 -ef file2 – stesso file`

`man bash` `per una lista completa`

Operatori di Input/Output

- Operatori di I/O
  - `|` - pipe, redirige lo stdout di un programma sull stdin di un altro
  - `>` - out su file
  - `>>` - appende a file
  - `<` - input da file

```
ls > ls.out
who >> who.log
```

Liste di comandi

- Per concatenare più comandi
  - `;` - necessario se sulla stessa riga di un file si vogliono inserire più comandi, eseguiti sequenzialmente e comunque
  - `&&` - operatore di corto circuito tra comandi (AND)
  - `||` - operatore di corto circuito tra comandi (OR)

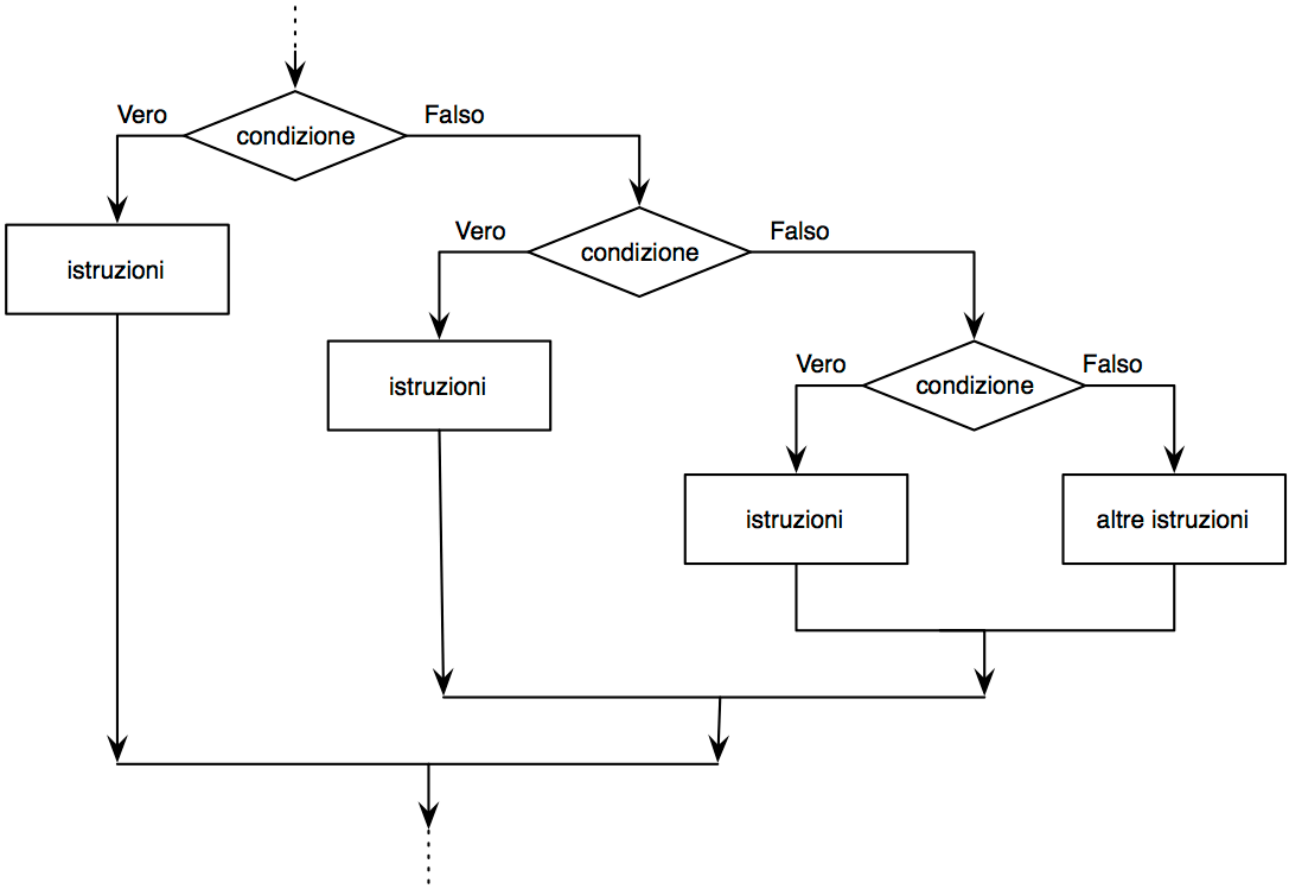
Esempio d'uso degli operatori

- `#!/bin/bash`

- `[ -e $HOME/fi1 -a ! -e $HOME/fi2 ] \ && echo “In $HOME c'è fi1 e non fi2”`
- `[ $HOME/fi3 -ef $HOME/fi4 ] && echo \ “fi3 e fi4 sono lo stesso file”`
- `cat $HOME/fi5 > $HOME/fi6`

- Ovviamente si possono usare strutture di controllo come gli if ...

Strutture di controllo



Cicli

- Per la costruzione di cicli, in bash si possono usare
  - **while** – simile al c
  - **until** – simile al while
  - **for** – semanticamente diverso dal c

While

- Esegue un blocco di codice finché una certa condizione è vera

```
while [ $ANS != “e” ]
do
    echo “Batti un tasto, con \”e\” esci”
    read ANS
done
```

O in alternativa: `while [ $ANS != “e” ] ; do`

Until

- Esegue codice finché una condizione è falsa

```
until [ -z “$1” ]
    Do echo –n “$1”
    Shift
Done
```

Lo `shift` “porta avanti” l’indice dei parametri posizionali

For 1/2

- La sintassi e semantica del for è diversa da altri linguaggi, come il c

```
for ELEMENT in LIST

do
    instructions

done
```

For 2/2

```
for planet in Mer Ven Ear Mar Jup

do
   echo $planet # Each planet on a separate
   line.

done
```

```
for i in $(seq 1 10)

do
   touch file$i; echo “file$i creato”

done
```

Istruzioni di selezione

- Le istruzioni di selezione disponibili sono
  - `if`
  - `if – else`
  - `if – elif – else`
  - `case`

La semantica di tali operazioni è del tutto analoga al c

If & co

```
read ANS


if [ $ANS == “y” ] ; then
   echo “you pressed y”
elif [ $ANS == “n” ] ; then
   echo “You pressed n”
else
   echo “Press y or n”
fi
```

Case

```
read ANS

case $ANS in
   y|yes)
       echo "you have sayed yes"
       ;;
   n|no)
       echo "You have sayed no"
       ;;
   *)
       echo "Press y, yes, n or no"
       ;;

esac
```

Opzioni agli script

Opzioni 1/2

- Per lavorare più agevolmente con le opzioni da passare ad uno script sono previste delle funzioni specifiche
  - getopts STRINGA_OPZIONI NOME
  - Unitamente a $OPTARG, variabile d'ambiente che prende il valore dell'argomento dell'opzione attualmente considerata

Opzioni 2/2

```
while getopts ":ab:c" OPTION; do
    case $OPTION in
        a)
            do something
            ;;
        b)
            VAR_FOR_b=$OPTARG
            do something
            ;;
        c)
            do something
            ;;
        ?)
            echo "Wrong option"
            ;;
    esac

done
```

Altri argomenti

- Espressioni regolari
- Linguaggi di scripting alternativi (python, perl, php, …)
- …
- …

Riferimenti

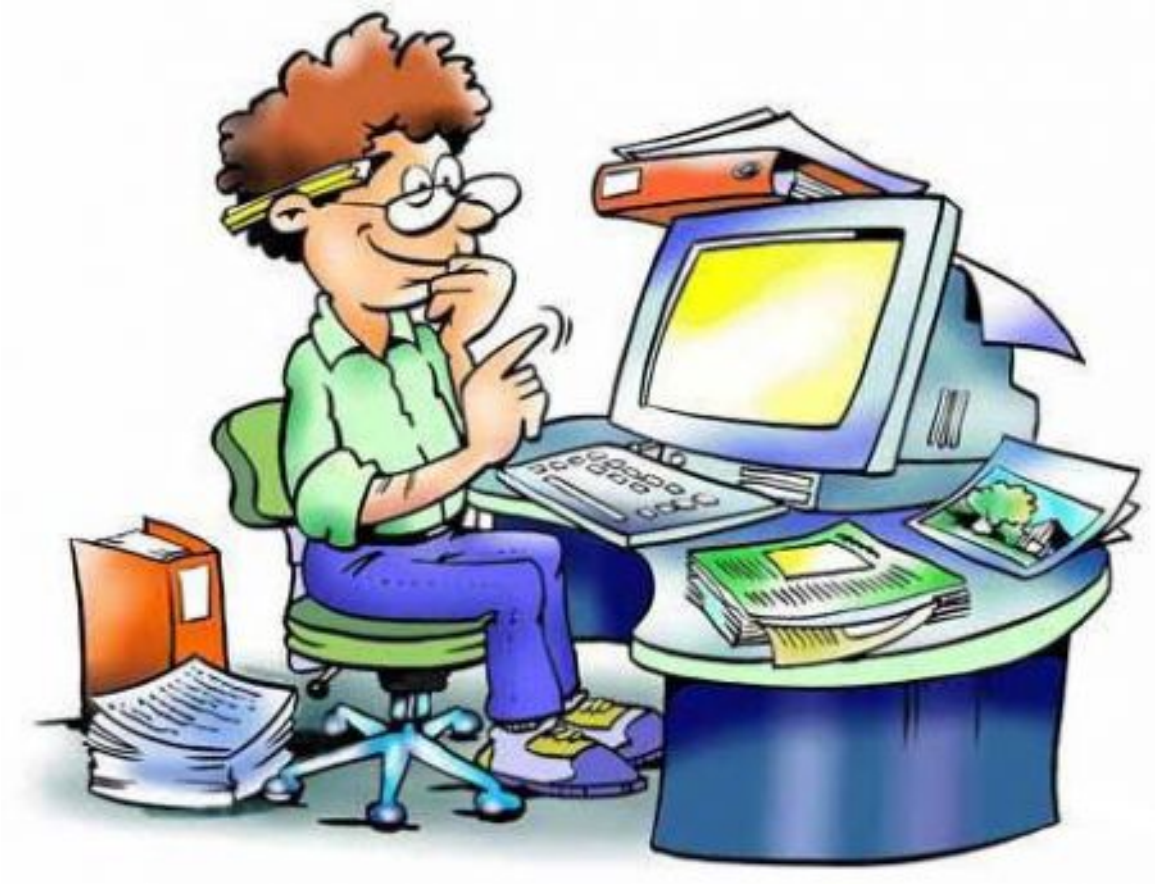
Advance Bash-Script Guide

http://www.tldp.org/LDP/abs/abs-guide.pdf

Unix Power Tools, 3rd Edition

Shelley Powers, Jerry Peek, Tim O'Reilly, Mike Loukides

Esercizi



Esercizi 1/3

- Scrivere uno script bash che provveda a controllare periodicamente ogni 30 secondi se l'utente passatogli come argomento abbia eseguito il logout
- Scrivere uno script che provveda allo svuotamento della directory D passatagli come argomento. Per ogni file contenuto lo script chieda conferma dell'eliminazione all'utente o spostamento in altra directory S data. Alla fine cancelli la directory D.

Esercizi 2/3

- Scrivere uno script che conti il numero di file non leggibili nella cartella home dell'utente
  SUGGERIMENTO: find […] -perm […]
- Scrivere uno script che mostri il nome più lungo e più breve presente nel sistema (/etc/passwd primo campo)
  SUGGERIMENTO: size=${#myvar}

Esercizi 3/3

- Scrivere uno script di nome copiafile che preveda la sintassi
  - copiafile Origine Dest

  Dove Origine e Dest sono nomi assoluti di directory. Lo script deve ricercare tutti i file ordinari contenuti in Origine e sottocartelle.

  Il file F di volta in volta trovato se solo leggibile va copiato nella cartella “readonly” all'interno di Dest, altrimenti in “altri”. Nel caso in cui più file con lo stesso nome vadano copiati nella stessa cartella rinominarli per evitare sovrascritture.

Soluzioni

```
#!/bin/sh
user=$1
while : ; do
    who | grep $user &> /dev/null
    if [ $? -ne 0 ]; then
        break
    else
        sleep 30
        continue
    fi
done
echo "$user has logged out at `date`"
```

Gestione software in Debian

deb, dpkg, apt

Fabio Del Vigna - fabio.delvigna@for.unipi.it

Sommario

- Installazione SW in Unix
- Pacchetti Deb
- Frontend per installazione:
  - dpkg, apt, aptitude

Introduzione

Installazione SW in Unix

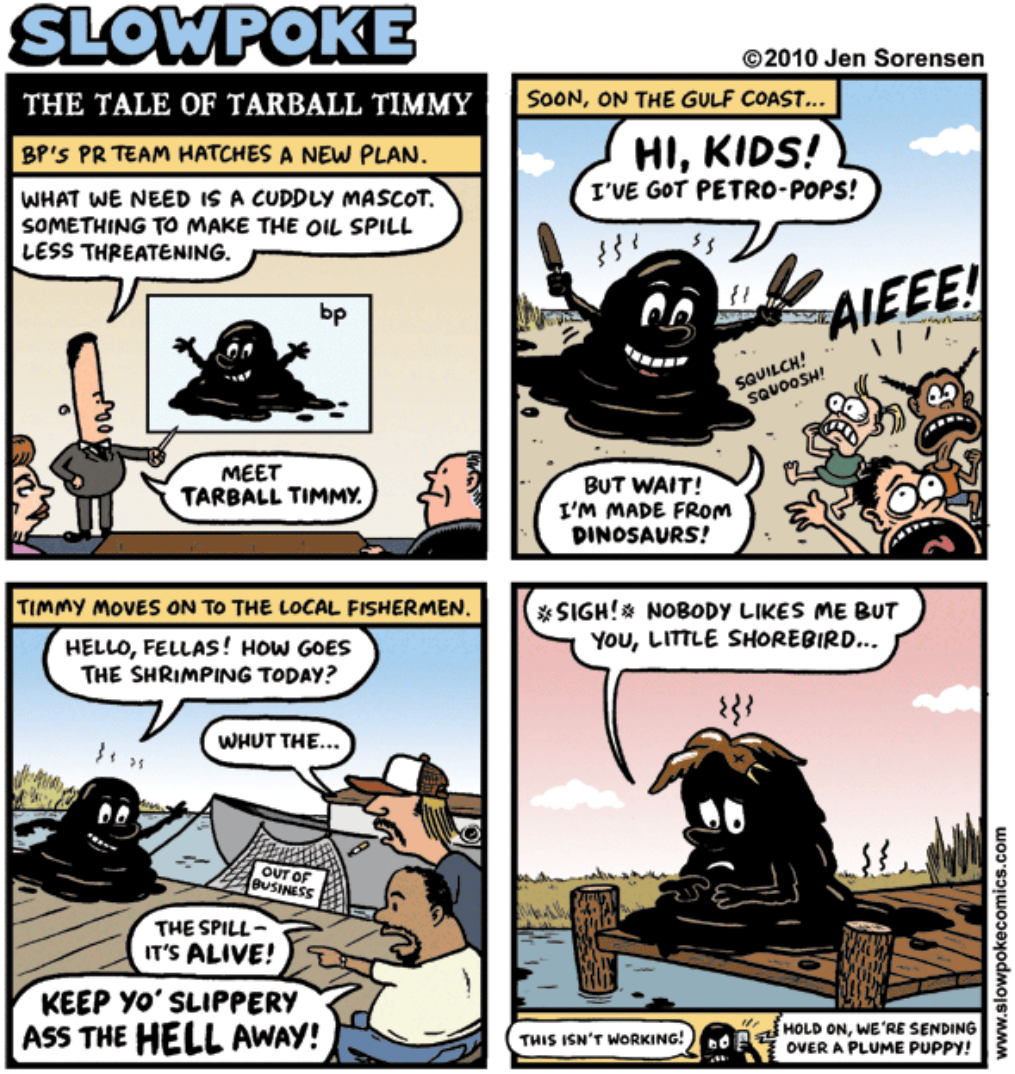
Fabio Del Vigna - fabio.delvigna@for.unipi.it

Introduzione

- In Debian si trovano alcune applicazioni base. Nel caso non fossero sufficienti è possibile installare software di terze parti nel sistema.
- Per l'installazione di software nei sistemi Unix è in generale necessario scaricare un archivio (***tarball***), decomprimerlo, configurarlo, compilarlo e installarlo sul sistema.
- In Debian esistono alternative all'installazione manuale del software: i pacchetti precompilati (.deb) e i pacchetti sorgenti

Tarball

1) Tarball: a blob of petroleum which has been weathered after floating in the ocean.



Tarball

1) Tarball: a blob of petroleum which has been weather after floating in the an.

NO!

Tarball YES

Tarball: An archive, created with the Unix tar(1) utility, containing myriad related files. “Here, I'll just ftp you a tarball of the whole project.” Tarballs have been the standard way to ship around source-code distributions since the mid-1980s; in retrospect it seems odd that this term did not enter common usage until the late 1990s.

The Jargon File

Installazione di SW in Unix

(variante "estesa")

1) Scaricare il SW, codice sorgente o binario
2) Scompattare il SW (spesso un tarball compresso con `gzip` o `bzip2`)
3) Trovare la documentazione (file `INSTALL` o `README` o file in dir "`doc/`") e capire come installare il SW
4) Se il SW è sotto forma di sorgente compilarlo. Eventualmente bisogna prima modificare un `Makefile` o eseguire uno script `configure`.
5) Installare e testare il SW

Se non funziona, bisogna correggere il codice (se disponibile) e / o effettuare il porting sul sistema che si usa.

Pacchetti deb (precompilati)

- Nei pacchetti si trova una **copia precompilata** di tutti i comandi, file di configurazione, documentazione di una applicazione.

Pacchetti sorgente

Un pacchetto sorgente di Debian è un **insieme di file predisposti per automatizzare** il processo di **compilazione** dell'applicazione dal codice sorgente

Pacchetti precompilati

- Debian si occupa di risolvere le **dipendenze** sia nel caso si usino i package che si usi il meccanismo dei port.
- L'uso dei pacchetti precompilati è consigliabile, però volendo… si possono usare pacchetti sorgente.

Pacchetti vs Sorgenti

- Tarball più piccolo (in genere)
- Non richiedono compilazione aggiuntiva
- Non richiedono la conoscenza del processo di compilazione in Debian

Opzioni compilazione (es. tipo processore)

Alcuni SW sono distribuibili solo come sorgente (per licenza)

Col sorgente si può controllare il codice

Applicazione patch

Ad alcuni piace avere il codice sorgente per poterlo leggere, hackerarlo, prenderne spunto, …

Pacchetti

Installazione pacchetti precompilati

- `dpkg -i` → installa un pacchetto Debian da un file locale
  - Scarico il .deb che mi interessa con un browser o con wget (strumento batch per il fetch di risorse in rete)
  - `$dpkg -i bash-completion.deb` → installa pacchetto
- `apt-get install bash-completion` → scarica il pacchetto da un server in rete (specificato nel file /etc/apt/sources.list

Gestione pacchetti aptitude

- `aptitude` → elenca e descrive i pacchetti installati sul sistema (interattivo)
- `aptitude install nome_pacchetto` → installa un pacchetto nel sistema
- `aptitude show nome_pacchetto` → da informazioni su uno specifico pacchetto
- `aptitude remove` → rimuove un pacchetto

Gestione pacchetti apt

- `apt-get install nome_pacchetto` → installa un pacchetto nel sistema
- `apt-get download nome_pacchetto` → scarica il pacchetto
- `apt-get remove` → rimuove un pacchetto
- `apt-get source nome_pacchetto` → scarica i sorgenti ("a la Debian") del pacchetto (e li pone nella cartella corrente)

Gestione pacchetti configurare le sorgenti

- `/etc/apt/sources.list` → configuriamo i repository da cui attingere i pacchetti

```
deb http://ftp.br.debian.org/debian squeeze main

deb-src http://ftp.br.debian.org/debian squeeze main


deb http://ftp.br.debian.org/debian squeeze-updates main

deb-src http://ftp.br.debian.org/debian squeeze-updates main


deb http://security.debian.org/ squeeze/updates main

deb-src http://security.debian.org/ squeeze/updates main
```

Riferimenti

- Bash reference guide
  - http://tldp.org/LDP/abs/html/
  - http://www.gnu.org/software/bash/manual/bashref.html
- http://www.debian.org
- http://www.debianizzati.org

MAKE

make

```
make hello
```

- Si creano programmi a partire dal codice sorgente.
- Make esegue i passi necessari (compilazione, eventualmente altro) sui file sorgenti

```
make hello → cc -o hello hello.c
```

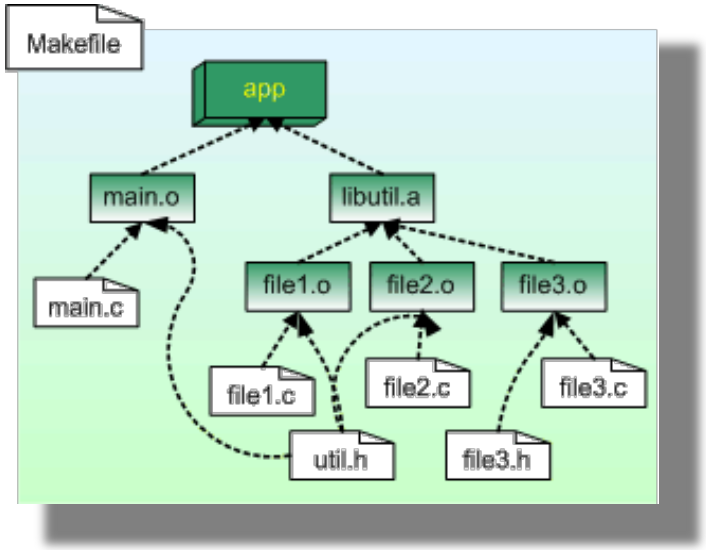
Makefile 1/2

- Per definire il comportamento di make si scrivono dei file (chiamati makefile).
- Nei makefile si definiscono delle regole
- Se non sono definite esplicitamente, make cerca di utilizzare regole di default

- Nota: i makefile hanno come nome standard makefile e Makefile

Makefile 2/2

- Nei makefile si indicano le interdipendenze dei file sorgenti e i comandi necessari per compilarli
- **Make esamina l'età dei file, e ricrea i file dipendenti da file modificati successivamente alla loro creazione.**



Regole di Make

- I makefile sono una serie di regole e macro, per creare gli obiettivi
- La sintassi base delle regole è

```
obiettivo { obiettivo } : [
  dipendenza ] [; comando] {<tab>
  comando }
```

  - L'obiettivo deve iniziare dalla prima colonna.
  - Ogni comando successivo deve essere rientrato con una tabulazione

```
program: program.o
    cc -o program program.o -lmylib
```

Macro

- Alcune Macro di default:
  - CC gcc
  - MAKE make
  - CFLAGS -0
  - LDFLAGS [vuoto]

- Tali macro possono essere ridefinite, con variabili di ambiente, o usando l'opzione -e [variabile] da passare a make

Macro - esempi

```
CC = g++                    # Per il C++
CC = gcc                    # Per il C
CFLAGS  = -g –Wall

CC = gcc
CFLAGS  = -g –Wall
TARGET  = myprog
all: $(TARGET)
$(TARGET):  $(TARGET).c
        $(CC) $(CFLAGS) -o $(TARGET)  $(TARGET).c
clean:
        $(RM) $(TARGET)
```

Regole di default di make

- Regole a un solo suffisso
  - `.c → $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $<`
  - `.sh → cp $< $@ ; chmod a+x $@`
- Regole a due suffissi
  - `.c.a → $(CC) $(CFLAGS) -c $< $(AR) \ $(ARFLAGS) $@ $*.o ; rm -f $*.o`
  - `.c.o → $(CC) $(CFLAGS) -c $<`
- $@ è l'obiettivo della specifica regola.
- $< indica il nome del file che ha determinato la scelta della regola

Obiettivi ricorrenti

- Obiettivi "ricorrenti" dei makefile:
  - `clean:`
    ```
    rm -f *.o
    ```
  - `all: $(TARGETS)`
  - `clobber:`
    ```
    rm -f $(TARGETS)
    ```
  - `install:`
    ```
    [operazioni per l'installazione nel sistema dei
       file necessari per l'esecuzione del
       programma]
    ```

In TARGETS si mettono tutti gli oggetti da creare col make.

(es. TARGETS = first_exec second_exec third_exec)

Esempio di makefile

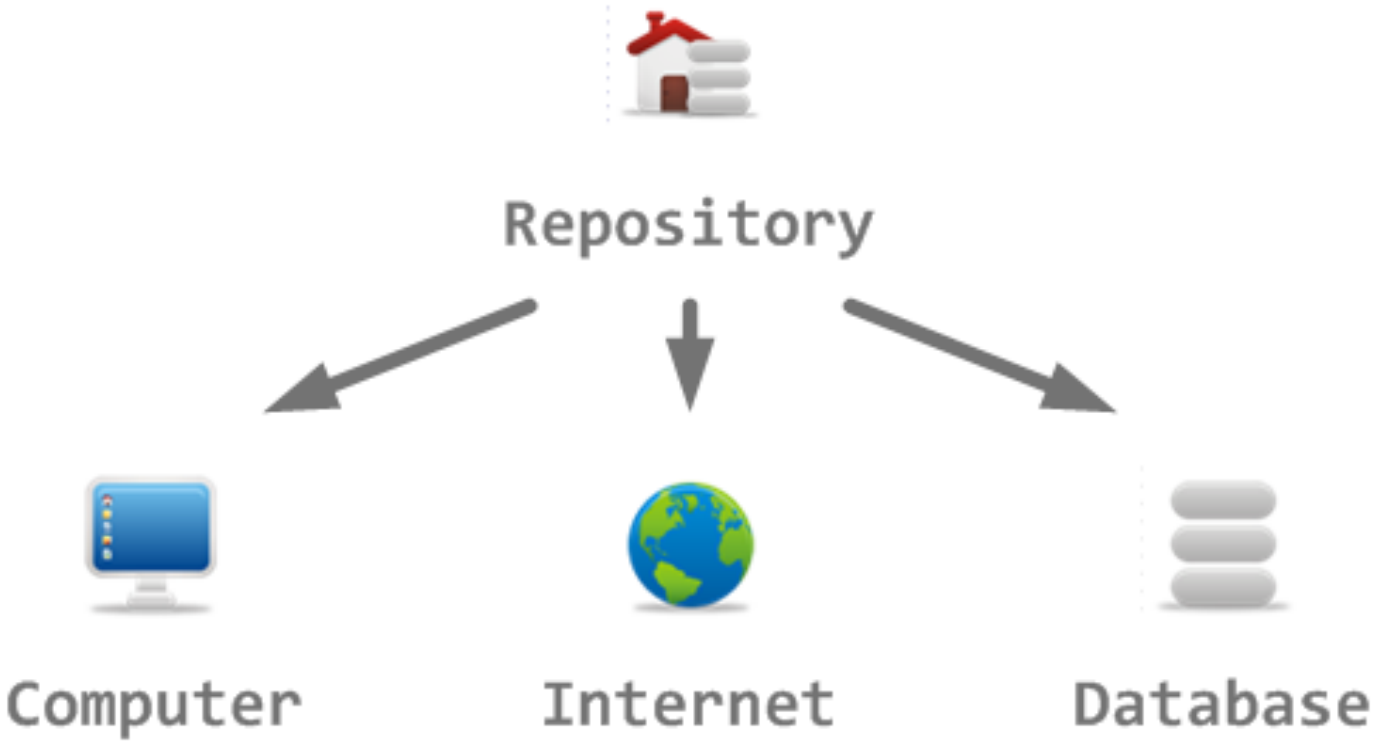
```
#Questo è un makefile

CC=gcc
CFLAGS=-Wall -Werr

main: main.o
  cc -o main main.o -lpthread

clean:
  rm -f main main.o
```

Repository



SVN - Introduzione

- SVN (SubVersioN) è uno strumento di “source control” o “revision control”.
- Tener traccia di modifiche a codice sorgente.
- Permette agli utenti di essere sincronizzati sui cambiamenti al codice.

Repository 1/2

- SVN salva tutti i dati tenuti sotto controllo in un apposito repository.
- Per usare SVN:
  - creare il repository

```
mkdir svnroot
svnadmin create
  svnroot/nome_progetto
```

- Il contenuto della cartella non va modificato manualmente.

Repository 2/2

- Dare i permessi di scrittura sulla cartella agli utenti che debbono usare quel repository.
- Esempio:
  - creare un apposito gruppo, svnusers
  - assegnare il repository a quel gruppo,
  - renderlo scrivibile dal gruppo e aggiungere gli utenti al gruppo svnusers

Eseguire il checkout

```
svn co [protocol://][user@][server][/repo]
```

- checkout di un intero repository SVN.
- Localmente crea una cartella che contiene una copia dei file scaricati dal repository.

Comandi svn 1

- `svn diff` → differenze tra il file modificato e l'ultima versione scaricata
- `svn ci` → commit delle modifiche eseguite localmente
- `svn up` → esegue l'update dei file dal repository verso la copia locale
- `svn help` → aiuto!!!

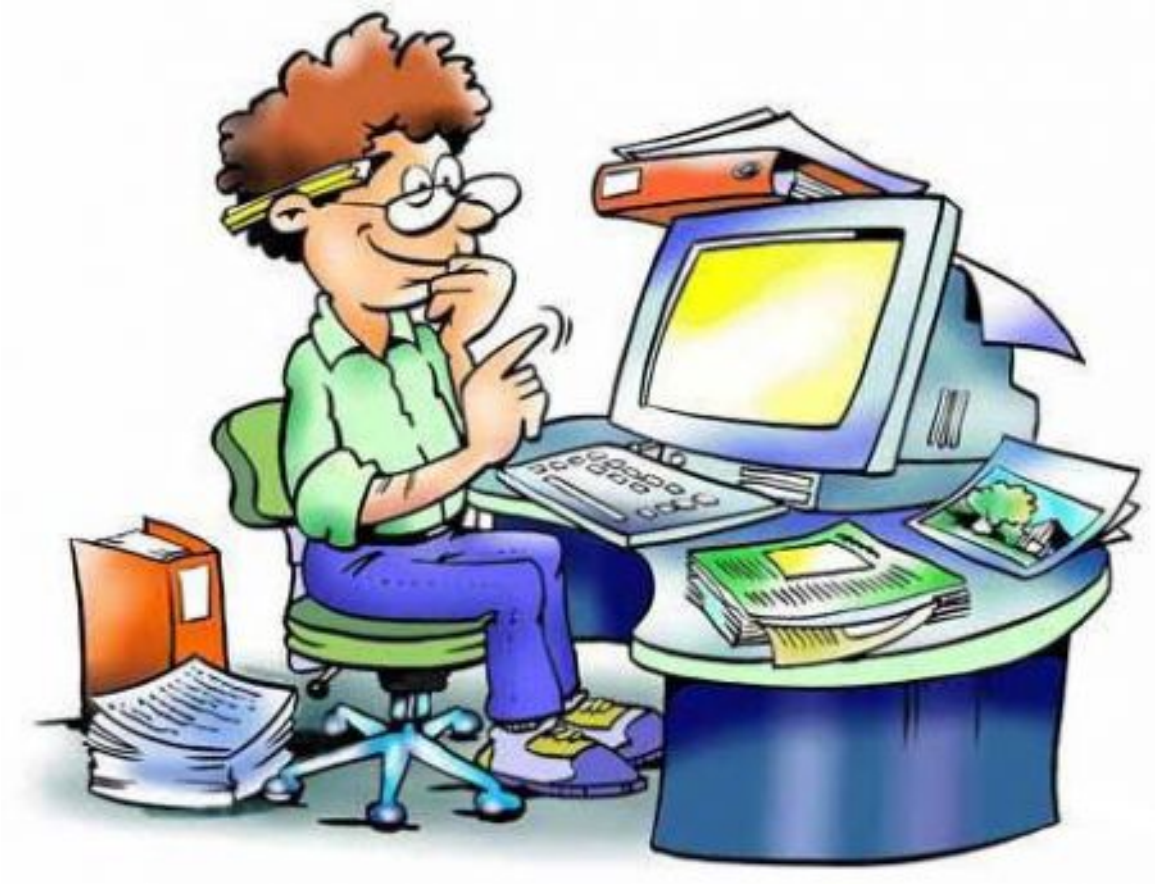
- Si possono invocare sia sull'intera cartella che su singoli file o sottocartelle

Comandi svn 2

- `svn add nomefile` → aggiunge un file
- `svn mkdir dirname` → crea una nuova cartella
- `svn delete name` → elimina il file o directory
- `svn move sorg dest` → sposta "sorg" in "dest"

- Attenzione: ciò che è sotto il controllo di SVN va alterato solo con i comandi appositi

Esercizi



Esercizi 1

- Leggere i makefile contenuti nella cartella del codice
- Scrivere un file sorgente in c (.c) e un file header (.h) incluso dal file .c. Compilare il file.c con gcc. Scrivere poi un makefile per la compilazione del file .c, specificando le dipendenze. Provare a modificare il file .h (ponendoci ad esempio una costante, che viene stampata a video dal programma). Modificando il file.h cosa succede durante la compilazione con make? E cosa succede se non è indicato nel makefile la dipendenza del file.c dal file.h?

Riferimenti

- *GNU Make: A Program for Directing Recompilation* by Richard M. Stallman and Roland McGrath
- *http://www.x.org/* - Official site of the Xorg foundation
- *http://www.linfo.org/x.html* – introduction to X
- *http://tools.ietf.org/html/rfc1198* – RFC

Riferimenti SVN

Controllo di Versione con Subversion: http://svnbook.red-bean.com/ (disponibile online e pubblicato da O'Really)