

# Corso formazione ReteIDRA

Savona 23/2 – 30/4 2010  
corso avanzato

Alessandro Dentella



# Lezione 2

- Ripasso
- Una premessa sulle reti per compilare i dati
- Uso editor (mimale)
- Mettere assieme i comandi: la bash
- I file di configurazione
- Hg: un modo di tenere traccia delle modifiche alla configurazione



# Comandi visti

- Cosa sono le opzioni?
  - Cosa sono gli argomenti?
  - cp
  - ssh
  - scp
  - man
  - ls
  - cat
  - rm
  - less (come si esce?)
  - WC
- Operatori:
- > *nome\_file* scrivi il risultato sul file *nome\_file*
  - >> *nome\_file* accoda il risultato a *nome\_file*
  - | passa l' output da un programma all' altro
- `dpkg -l | wc -l` (conta i pacchetti installati)



# operatori

- I comandi appena introdotti producono un output che può essere immesso in un comando successivo o scritto su file con alcuni operatori:
  - | (pipeline). Potreste contare i pacchetti così:  
`dpkg -l | wc -l`
  - > scrivi l' output nel file:  
`dpkg -l > /tmp/pacchetti_installati`
  - >> accoda l'output ad un file esistente  
`cat id_dsa.pub >> .ssh/authorized_keys`



# generazione chiavi

- Per generare le chiavi sotto linux occorre lanciare il comando:

```
ssh-keygen -t dsa
```

- Il comando genera una coppia di file pubblico/privato. Quello privato non deve essere dato a nessuno, quello pubblico può essere anche messo su un server web



# authorized\_keys

- Per potere accedere senza chiave ad un server occorre che questo abbia formalmente accettato la nostra identità fra quelle autorizzate: occorre mettere la nostra chiave pubblica nel file `authorized_keys` del server a cui vogliamo collegarci:

```
scp .ssh/id_dsa.pub user@srv_remoto:/tmp  
ssh user@srv_remoto 'cat /tmp/id_ds.pub  
>> .ssh/authorized_keys'
```

- La seconda riga accoda il file appena inviato al file `authorized_keys` (>>)
- In entrambe le righe sopra riportate ci viene chiesta la password




... sostituire `srv_remoto` e `user`... ovviamente!

# Ssh da chiavetta USB

- Abbiamo visto che è possibile usare il comando ssh facendo pescare la chiave da un file usando l'opzione -i
- Come di può usare ssh da chiavetta?
- Se serve da windows, potete mettere chiave ssh e putty sulla chiavetta USB, protetta da password



# Studio pacchetti installati

- Potete vedere una presentazione [qui](#)
  - repository in /etc/apt/sources.list
  - apt-get update
  - apt-get upgrade
  - apt-get install nome\_pkg
  - aptitude
  - apt-get remove --purge
  - dpkg -S nome\_file
  -  dpkg -I pkg

# sources

- I pacchetti vengono prelevati da alcune sorgenti normalmente via http. Le sorgenti sono suddivise per architettura (i386, amd64) e per codename (lenny, etch...) e component (main, free...)
- Ogni sorgente mette a disposizione un file **Packages** (firmato) con l'elenco dei pacchetti forniti dal repository.



- Un pacchetto potrebbe essere disponibile per

# dipendenze

- I pacchetti sono strutturati in un albero di dipendenze. Se isi dipende da python-isi, non sarà possibile installare isi se non è stato (già) installato python-isi.
- Apt-get è il comando che installa, ovvero informa il sistema di prelevare dalla sorgente che ha la versione più recente e di darlo a dpkg per la installazione in locale.
- Se apt-get vede che il pacchetto richiesto dipende da un altro pacchetto, procede all'installazione di tutte le

# Dipendenze non soddisfatte

- A volte le dipendenze non possono essere soddisfatte, normalmente per errore di chi ha creato il pacchetto o perchè le sorgenti non sono complete
- Forzare le dipendenze è una operazione che va fatta solo se si è sicuri che la cosa non comporti problemi
- Il comando 'dpkg -l' mostra tutti i pacchetti installati e la loro versione
- Esercizio: vediamo tutti i pacchetti installati sul sistema

 Vediamo tutti i pacchetti con nome isi  
*lara*

# Trovare file

- Ci sono vari modi di cercare un file, ognuno ha il suo comando:
  - Dove sta il comando “ls”: `which ls`  
il sistema cerca nel PATH per capire dove risiede il file usato
  - Quale pacchetto ha installato il comando “ls”?  
`dpkg -S $(which ls)`
  - Qui `$( )` significa: sostituisci qui il risultato di “which ls”. E` abbastanza probabile trovarlo nella documentazione



idra

# Find & locate


- find: trova un file con certe caratteristiche cercando nel filesystem
- locate: trova un file con un certo nome cercando in un database



# FHS (seconda parte)

- I programmi devono essere installati in `/usr/*` o in `/opt` a seconda della struttura interna. I gestori li metteranno in `/usr`, gli installer dei programmi possono mettere in `/opt`
- `/var` è una cartella per dati rapidamente variabili (log, database, mail, ldap, proxy web, cache in genere...)
- `/tmp` è per file temporanei
- `/root` per la home del superuser

# /etc

- Di gran lunga la cartella più interessante ed importante del sistema, /etc ha tutte le configurazioni ad eccezione di:
  - Configurazioni individuali (nella home)
  - Database Idap e database in genere se usati
- Molti file di configurazione sono leggibili da tutti, solo le password sono normalmente protette
-  Esercizio: analizzare come è strutturata la cartella di configurazione di isi: dove sta?

# /home

- /home è la cartella dove vengono conservati tutti i dati individuali
- Non ci sono solo le home degli utenti, c'è ogni cartella da condividere. Normalmente è strutturata come partizione separata: come mai?



# Un comando importante: tar

- Simile al comando zip di windows: che cosa fa?

Esempi di uso:

```
tar cvf ~/etc.tar /etc
```

```
tar cvf ~/isi.tar /etc/isi
```

```
tar xzvf ~/isi.tgz
```

- [ ~ significa \$HOME, la mia cartella ]



# tar

- Il compito di tar è di riunire molti file in un unico file
- Spesso è usato in congiunzione con un sistema di compressione: gzip o bunzip2 per i quali ha due opzioni:

x: gzip (meno potente, più veloce)

j: bunzip (più potente, più lento)



# tar

- Il comando tar viene usato molto: crea un file archivio: un file che contiene un'intera collezione di file e cartelle al pari di zip.
- Questo comando accetta che le opzioni **non** abbiano il -, ad esempio per scompattare:
- `Tar xzvf /tmp/backup/etc.tar.gz`



# Esercizi

- Usare `tar` per creare una copia di backup di `/etc` e metterla in `/tmp/$user-etc.tgz` dove `$user` è il vostro nome.
- Scompattare lo stesso file e cercare se contiene un file chiamato
  - `Slapcat`
  - `logon.conf`
- Se li trovate contate quante righe hanno



# Editor

- L' editor è un programma che ci permette di modificare il contenuto dei file
- Un editor può fare molte cose (il mio, emacs, ha quasi 1000 funzioni) ed al minimo deve permettere di:
  - Editare: `jmacs pippo.txt` (`nano pippo.txt`)
  - Salvare: `control-x control-s` (`nano: Control-o`)
  - Uscire: `control-x control-q` (`nano: Control-x`)



# Editor + diff

- Contate quanti file ci sono in /etc a partire dalla copia di backup
- Contate quanti ce ne sono usando find
- Modificare un file con un editor (jmacs o nano – nell'allegato) e verificare con il comando diff come il sistema vede le modifiche:

```
cp /etc/fstab /tmp
```

```
jmacs /tmp/fstab
```

```
 diff /etc/fstab /tmp
```

# Reti

**kit minimo di sopravvivenza**




# Indirizzi IP

- Gli indirizzi ip funzionalmente hanno una stretta analogia con i numeri telefonici, cambia la forma:

192.168.1.13

335/838.8383

- Avere due numeri di telefono, rende troppo raggiungibili, problema di sovrapposizione
-  Se parlassimo con una persona da un telefono e ci riprendesse da un altro (fisso/cell)

# locale/non locale

- Le telefonate locali sono (forse erano...) quelle con identico prefisso
- Analogamente gli apparati con uguale inizio sono sulla stessa rete locale. Quanti numeri debbano essere uguali lo vedremo la volta prossima:

192.168.1.13

192.168.1.254

■  192.168.2.15??

# Anche i pc!

- Analogamente anche i pc se hanno due IP (sulla stessa rete) ma su due interfacce differenti, trovano problemi:
  - Non sanno quale usare
  - Ne scelgono una che può essere differente da

accesso

Eth0: 192.168.0.4 Eth1: 192.168.0.253 Gw: 192.168.0.254

Raggiungibile: Da ip esterno su porta ssh Password key: SI



# gateway

- Cosa hanno in comune, un telefono tradizionale quando staccate il filo ed un telefono cellulare in mezzo al deserto?
- Non sanno come uscire. Gli manca il punto per uscire ed andare nel mondo
- Se non vanno nel mondo il mondo non li raggiunge

accesso

```
Eth0:          192.168.0.4          Eth1:          192.168.0.253 Gw: 192.168.0.254  
Raggiungibile: Da ip esterno su porta ssh Password key: SI
```



# La shell bash

*il collante che dai mattoncini ci permette di creare un programmino*



# scopo

- La shell permette di creare veri e propri programmini che aiutano ad automatizzare compiti di routine
- Tutta reteisi è nata con script di shell (ora quasi tutto python)
- Esempio: automatizziamo l'aggiunta della chiave:
  - Prelevandola da internet
  - Accodandola al file `ssh/authorized_keys`



*idra*

# Studiamo cosa esiste


- Esiste `add_key`: dove sta?
- Come leggerlo?



# intestazione

- Per scrivere una script di shell occorre
- `#!/bin/bash`  
intestazione: informa il sistema che l' interprete per questi comandi è la shell `/bin/bash`
- Comandi come li daremmo da riga di comando, abbiamo anche a disposizione `if`, `for`, `while` e `case`:

```
if [ -f /etc/resolve.conf ] ; then
```



*idea*  
echo "Si c'è"

# If nella bash

- Notare: if ; then fi (chiude if)
- test -f, per capire quali altri test possiamo fare:  
help test
- Le parentesi quadre richiedono spazi prima e dopo!!!!
- Se volete potete aggiungere:

else



# For nella bash

- Un ciclo potrebbe essere così:

```
for file in /etc/passwd /etc/shadow; do  
    cp -b $file /home/backup  
done
```

- Notare: for ; do ; done
- File è il nome della variabile che contiene i nomi dei file /etc/passwd e poi /etc/shadow
- Per chiamare una variabile usiamo il \$: \$file

■  Per assegnare una variabile:

# Esercizi

- Guardare i file in `/etc/init.d/*` e vedere se capiamo la sintassi: usano costrutto:

```
case $a in
    start) do_start
        ;;
    stop) do_stop
        ;;
esac
```

- Usano anche `.file` che forza la lettura del file come codice. E` un modo di inglobare altre

# Esercizi

- Creare una script che dica se esiste il file `/usr/share/netkit/fs/isi-lenny.img`, avvisare a schermo e con una mail (comando `mutt` o `mail`, quello che è presente)
- Creare una script che ritorni il pacchetto `debian` che contiene un comando dato come argomento
- Creare una script che “raddoppi” un file accodandolo due volte nell'output e che lo scriva



*idra*

# Argomenti in una script

- Gli argomenti di una script di shell vengono visti dall' interno come: \$1, \$2, \$3...

```
FILE=$1
```

```
if [ -f $FILE ] ; then echo ok; fi
```



# Altro esempio: vpn

- Installiamo una vpn fra la vostra scuola ed il mio ufficio. Serve
  - La configurazione
  - La chiave
  - Vanno messe in /etc/openvpn
  - Va avviato il servizio
  - Va configurato il firewall

```
..... tar xzvf idra_ovpn.tar.gz
```



*idra*

```
..... sh ovpn/installa_openvpn.sh
```

# Mercurial

**la nostra macchina del tempo**



# trailer

- Prendiamo una cartella, ci copiamo dei file. Voi li modificate a piacimento ed io vi dico cosa avete modificato.

```
cp -av /etc/isi ~
```



# /etc con mercurial

- mercurial è un DVCS (distributed version control system). Aiuta a tenere traccia delle modifiche ai file di configurazione
- mercurial nasce come strumento per i programmatori ma risulta comodo anche per i sistemisti

# VCS

- inizialmente rcs, cvs, svn. Il principio è che si tiene un repository centrale, ovvero una versione ufficiale di un codice e si permette di modificarlo a dei client.
- le modifiche si chiamano **commit** e salvano le differenze rispetto alla versione precedente
- alcuni comandi permettono di verificare se siamo allineati con la versione ufficiale o se ci sono delle differenze:



idea

# diff e patch

- Alla base di questi sistemi c'è il comando `diff` che permette di vedere quali differenze ci sono fra due versioni. Questo comando è utile indipendentemente dai sistemi CVS
- un file con le differenze può essere applicato ad un file originale con il comando `patch`

# dVCS

- I sistemi distribuiti aggiungono un livello: il repository dove si salvano le differenze con commit è sempre locale. Poi le differenze possono essere sincronizzate con un sistema remoto.
- Questo crea complicazioni e opportunità. Noi ci limitiamo ad usarlo in locale

# repository/working directory

- La creazione di un repository hg è facile come scrivere:

`hg init`

questo crea una cartella `.hg` nella quale sta il repository, ovvero i file con la storia di tutto il sistema. Quando usiamo `hg init` il sistema non contiene file

- `hg add` prepara l'aggiunta dei file al repository



*idra*

- `hg commit` li aggiunge realmente

# Sessione di lavoro

```
$ mkdir test
```

```
$ cd test
```

```
$ echo ciao > README
```

```
$ hg init
```

```
$ hg st
```

```
? README
```

```
$ hg add
```

```
adding README
```

```
$ hg st
```

```
A README
```

```
$ hg ci -m "primo commit"
```

```
$ hg st
```

```
$ hg log
```

```
changeset: 0:885d921d0718
```

```
tag:  tip
```

```
user: sandro
```

# hg commit

- hg commit NON modifica i file
- hg commit richiede un messaggio che possiamo passare con l'opzione -m
- accetta dei file come argomenti: in quel caso aggiunge solo questi file
- se sbagliamo, possiamo usare hg rollback



# hg status

- hg status ci dice quali file non sono sincronizzati con il repository (ovvero quelli che hanno una versione della working directory differente da quella del repository)
- accetta come argomento file e cartelle
- ci dice anche se sono differenti i permessi, ma li mostra solo con opzione -git



# hg log

- hg log mostra la storia delle modifiche di un file.  
È la ragione, assieme a diff e revert per cui si interessa il sistema di controllo versioni

# hg revert

- hg revert permette di ripristinare la situazione come era ad un certo punto (revisione). Es.:  
hg revert --rev 10

# hg diff

- chiede al sistema di mostrare le differenze fra due revisioni o fra la working directory ed una specifica revisione.

# hg cat

- hg cat permette di prendere un file di una revisione indicata con l'opzione -r e di metterlo sullo stdout