CST8177 – Linux II

ssh keys, yum, ntp, rsync

Todd Kelley
kelleyt@algonquincollege.com

Final Exam

- CST8177 Linux Operating Systems II
- Thurs 24-Apr-14 10:30-12:30 T119

Today's Topics

- ifconfig to find your VM's ip address so you can ssh to it
- ssh key login
- yum
- ntp
- tar
- scp
- rsync

IP address of your CentOS VM

- run the /sbin/ifconfig command
- on your new install, you'll have only your root account at first:

```
# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0C:29:14:F8:93
          inet addr:192.168.180.207  Bcast:192.168.180.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe14:f893/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1112 errors:1099 dropped:0 overruns:0 frame:0
          TX packets:4178 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:210424 (205.4 KiB)  TX bytes:624100 (609.4 KiB)
          Interrupt:19 Base address:0x2024
```

ssh key-based login

- key-based logins are more secure than password logins
- you run ssh to log in from a client to a server
- on the client, you have a private and public key pair (with passphrase)
- on the server, you put your public key into ~/.ssh/authorized_keys
- IMPORTANT: permissions 700 on `.ssh/`, 600 on `authorized_keys`
- when you log in from the client to the server, you're prompted for your key's passphrase

Key pair principle

- Anybody can generate a matching private/public key pair
- You let anybody and everybody have a copy of the public key – it's public!
- You keep your private key secret and hidden, only you have it – it's private!
- Anyone who has your public key can use your public key to create a challenge that only someone with the matching private key can meet (in other words, only you can meet).

Key pair principle (cont'd)

- Key pairs are designed mathematically so that something encrypted with one key of the private/public pair can be decrypted by only the matching key of the pair.
- Something encrypted with the public key CANNOT be retrieved with the public key – the private key is required!
- So the "challenge" is to encrypt a message (maybe random) with the public key.
- Only someone with the private key can decrypt the message and retrieve the original message

Ssh key login process

- A copy of your public key is stored in authorized_keys in your account
- You have the private key and you tell the ssh server that you want to log in
- The ssh server says "OK", I have your public key here, and I'll use it to encrypt this random message, and give the result to you. If you can tell me the original message, you must have the private key (and you must be you).
- You use your private key to retrieve the original message, and send it back to the server.
- The server lets you in, because it assumes only you have the private key necessary to retrieve the original (random) message.

key pair analogy

- Cinderella's slipper?
  - the assumption is that only Cinderella's foot (which only Cinderella has) will fit into the slipper
  - The foot and the slipper match like a private and public key match
  - One difference is that in public key cryptography, everybody has a copy of the slipper (only the real Cinderella has the foot that will fit the slipper)
  - If I want you to prove to me that you are Cinderella, I get you to prove to me that your foot fits in the Cinderella slipper – if it fits, you must be Cinderella
  - The mathematics ensures that nobody else's foot will fit somebody's slipper, and given a slipper, you cannot create the matching foot.

ssh key login Linux

- Generating a keypair on Linux client:

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/tgk/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/tgk/.ssh/id_rsa.
Your public key has been saved in /home/tgk/.ssh/id_rsa.pub.
The key fingerprint is:
81:27:65:81:26:fb:1b:6c:71:ae:a0:9c:58:5b:64:3b tgk@localhost.localdomain
The key's randomart image is:
+--[ RSA 2048]----+
|      .+.        |
|    . o+         |
|     +o o        |
|    + .o..       |
|   o + +S        |
|  . E = .        |
| + = + +         |
|. =   o          |
|                 |
+-----------------+
[tgk@localhost ~]$
```

ssh key login Linux (cont'd)

- install your (client) public key to the server
  - put the contents of `id_rsa.pub` (we generated this file on the client) into `~/.ssh/authorized_keys` on the server
  - can do this with vi, copy-paste
  - alternatively, can do this with `ssh-copy-id` command
  - you're running this command on the client

```
client$ ssh-copy-id username@example.com
```

  - now you should be able to log in with the key, and you'll need to give your passphrase for your key

ssh key login Windows

- http://www.howtoforge.com/ssh_key_based_logins_putty

Yum: Yellowdog Updater Modified

- http://teaching.idallen.com/cst8207/14w/notes/520_package_management.html
- yum can install software packages for you, retrieving them from a repository over the network
- performs dependency analysis: if the package you want to install depends on another package, it will install that too
- can also query installed packages, remove packages, update packages, etc
- run with root privileges

Yum (cont'd)

- Examples: (see "man yum" for details)
  - yum install ntp
    - install the package "ntp" and its dependencies
  - yum update
    - update all currently installed packages
  - yum update "nt*" # quote the glob from the shell
    - update all packages that match the glob
  - yum -v repolist # print info about repositories
  - yum list installed # list the installed packages
  - yum list available # list the available packages
  - yum list # combination of two above
  - yum search fortune # search package names for fortune

Yum repository configuration

- we shouldn't need to change these, but if you're curious...
- repository files are in /etc/yum.repos.d
  - CentOS-Base.repo
    - main CentOS repository mirrors
  - CentOS-Media.repo
    - uses the DVD in your drive as a repository
- To configure your machine to use the EPEL repository (for cowsay, fortune-mod, etc):
  - rpm -Uvh http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm

NTP: network time protocol

- we'll be using the ntp package to keep our CentOS clocks synchronized with a time server, such as 1.centos.pool.ntp.org
- ntpd, the ntp daemon, will look after keeping our clocks accurate
- /etc/ntp.conf configures the daemon, and all we need to do is arrange for the daemon to start:

```
bash$ chkconfig ntpd on
bash$ chkconfig –list ntpd
```

NTP: cont'd

- now that the ntpd daemon is configured to start upon entering runlevels 2,3,4,and 5, let's check whether it's running:

bash$ service ntpd status

ntpd is stopped

- we are in runlevel 3 but we haven't actually entered that runlevel since we ran chkconfig
- we'll start it manually this one time:

bash$ service ntpd start

tar command basics

- create an archive of a directory
  - tar cvzf mydirectory.tgz mydirectory
    - c: create an archive
    - v: verbose, print the filenames as their added
    - z: compress the archive
    - f: use the following as the filename for the archive
- extract an archive
  - tar xvzf mydirectory.tgz
    - x: extract an archive
    - z: uncompress the archive

tar command basics (cont'd)

- print listing of an archive without extracting
  - tar tvzf mydirectory.tgz mydirectory
    - t: print a listing
    - v: verbose, like a long listing
    - z: the archive is compressed
    - f: use the following as the filename for the archive
- In each of the above examples
  - exactly one of t, c, or x is mandatory
  - f with an archive name is mandatory
  - z: is mandatory if archive is, or is to be, compressed
  - v: is optional for verbosity

Copying over SSH: scp

- scp behaves much like the familiar cp command, but with remote capabilities
- The arguments (source or destination) can optionally be for a remote file/directory
- http://teaching.idallen.com/cst8207/14w/notes/015_file_transfer.html
- A remote argument has a colon in it
- To copy local passwd file to kelleyt's home directory on a remote computer
  - `scp /etc/passwd kelleyt@cst8177.idallen.ca:`
  - Notice the colon in the remote dest argument
  - file `~kelleyt/passwd` on `cst8177.idallen.ca` is created or if it already existed, it's overwritten

scp (cont'd)

- Whatever name follows the colon is relative to the home directory on the remote side (unless it's an absolute path and therefore not relative)
- use -p option to preserve timestamps, modes (analogous to -p with `cp` command)
- Use CAPITAL P option to specify a port
  - if you're at a McDonalds and you want to copy to myuser's home directory on the CLS:
  - scp -P 443 localfile.txt myuser@cst8177.idallen.ca:
  - again, notice the colon in the remote argument
  - notice that port option is -p for ssh, -P for scp

More scp examples

- absolute local to absolute remote file foo
  - scp -p /etc/passwd user@remote.com:/home/user/foo
- relative local file to absolute remote directory
  - scp -p myfile user@example.com:/home/user/
- directory and its contents to remote directory
  - scp -rp mydir user@example.com:somedir
- absolute remote file to local home dir
  - scp user@example.com:/etc/passwd ~
- relative remote file to current local dir
  - scp user@example.com:somedir/foo .

rsync basics

- rsync behaves similarly to scp
- only one rsync argument can be remote
- Example copy local (relative) to local (absolute):
- rsync –aHv adir /some/dir
  - a: archive mode, preserve permissions, timestamps, etc
  - H: preserve hard links
  - v: verbose
  - if "dir" exists, "/some/dir/adir" will result
  - if "dir" does not exist, "/some/dir" will be created and contain "adir", "/some/dir/adir" will result

rsync basics: Trailing slash

- be careful with a trailing slash on the source
- a trailing slash on source has special meaning: copy the contents of the directory
- these are the same
  - rsync -avH /src/foo /dst/
  - rsync -avH /src/foo/ /dst/foo
- copy contents of src directory to dst directory
  - rsync -avH /src/ /dst # /src/* in /dst/
- copy src directory to dst directory
  - rsync -avH /src /dst #end up with /dst/src

That's hard to remember?

- If you can't remember directory creation, and directory contents (trailing slash) versus directory itself, then use dot as the source directory for unambiguous usage
- All of the following will sync the local `dir` directory to make the remote `rdir` directory the same as `dir`, whether `rdir` already exists or not
- Use the same name (`dir`) on both sides if desired

```
rysnc -avH path/to/dir/.  user@remote:path/to/rdir
rysnc -avH path/to/dir/.  user@remote:path/to/rdir/.
rysnc -avH path/to/dir/./ user@remote:path/to/rdir
rysnc -avH path/to/dir/./ user@remote:path/to/rdir/.
```

rsync (cont'd)

- rsync can copy across the network

rsync –avH dir/. kelleyt@remote.example.com:dir

- that will copy/synchronize the local "dir" with the remote "dir" in kelleyt's home directory on the remote machine named “remote.example.com”
- notice the colon in the remote argument
- if you forget the colon, you do a local copy to a file with '@' in its name

rsync (cont'd)

- after the colon, you can specify a relative path (relative to the home directory) or an absolute path

rsync –av adir/. kelleyt@192.168.0.193:/etc/adir

- that example uses an absolute path at the destination end, and an IP address instead of a hostname

rsync (cont'd)

- the other direction works too

rsync kelleyt@192.168.0.193:/etc/passwd .

- that copies the remote file /etc/passwd to the current directory (.), resulting in ./passwd
- this time, we are not using archive mode
- this time, we are using an IP address instead of a fully qualified domain name

rsync (cont'd)

- rsync compares source and destination and minimizes the number of bytes that need to be copied to update the destination
- rsync algorithm is designed to transfer only the parts of a file that have changed
- notice the "speedup" in the summary when you use the "-v" option