CST8177 – Linux II

ssh keys, yum, ntp, rsync

Todd Kelley
kelleyt@algonquincollege.com

Final Exam

- CST8177 Linux Operating Systems II
- Mon 22-Apr-13 12:00 14:00 CA105A,B,C

Today's Topics

- ifconfig to find your VM's ip address so you can ssh to it
- ssh key login
- creating many new users
- passwd command examples vipw vigr visudo
- yum
- ntp
- tar ssh/rsync
- disks
- partitioning
- formatting filesystems mkfs
- /etc/fstab
- mounting filesystems mount command

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

- when you log in from the client to the server, you're prompted for your key's passphrase

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
- 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

creating users

- by default, useradd creates the new user's home directory
- the new home directory is populated with the contents of /etc/skel/
- shadow password suite configuration in /etc/login.defs
- the defaults for useradd are /etc/default/useradd

creating many new users (cont'd)

- to create one user:

useradd -c "Full Name" user001
chmod 750 /home/user001
passwd user001 # and enter passwd by hand

creating many new users (cont'd)

- there are various possible strategies for creating many new user accounts
- one possibility:
  - use Linux utilities and/or your own script to create a set of commands for each new user (one-off script):

```
useradd -c "User 1" user001              #create the user
usermod -p u75jjvrue5B92 user001  #assign passwd
chmod 750 /home/user001 || exit 1 #homedir perms
```

- If you were creating 100 users, you'd have 300 commands in your one-off script

creating many users (cont'd)

- another possibility: the "newusers" command
- man newusers
- newusers takes a file containing info about the accounts you want to create
- the input file for creating the accounts is in the same format as the /etc/passwd file:

uncle:3uncle4:503:503:Uncle Tom:/home/uncle:/bin/bash
aunt:3aunt4:504:504:Aunt Betty:/home/aunt:/bin/bash

passwd command examples

- bad idea: set blank password for user
  - passwd -d username # shouldn't need to do this
  - sets blank password field in /etc/shadow
  - login still prompts for password, so you'd need to jump through hoops to allow for login with blank password
  - su will not prompt for passwd

passwd examples (cont'd)

- disable passwd authentication for username
  - passwd –l username # puts ! in front of passwd
  - passwd –u username # undoes the above

- a ! placed in front of the passwd entry of the shadow file ensures that nothing anybody can type will successfully match this passwd entry
- * in the passwd entry is similar, and used for accounts for which should never use passwd authentication
- SSH keys will still work without passwd

passwd examples (cont'd)

- passwd –n mindays
- passwd –x maxdays
- passwd –w warndays
- passwd –i expireaccountdays
- example: allow changing password no more than once per day, force changing every 90 days, warning 10 days in advance of expiry, and if they don't change their password within 2 days after expiry, disable account (not even ssh key login will work):
- passwd –n 1 –x 90 –w 10 –i 2 username

force passwd change on login

- chage –d 0 username
- thereafter, the first time the user logs in, they will be forced to enter their password
- all the other aging parameters are unchanged (maxdays, lastday, mindays, etc)

Editing critical files

- Don't edit files when there's a command that updates the file
  - e.g. "usermod -c 'New User' newuser" instead of changing gecos field in /etc/passwd by hand
- If you must edit the file, don't edit it directly when there's a command for that purpose (vi will be the default editor):
  - visudo # edit the /etc/sudoers file
  - vipw # edit the /etc/passwd file
  - vigr # edit the /etc/group file
- normally can specify a different editor in EDITOR or VISUAL environment variables (see man)
- can set these in .bashrc, export them!
- Command line examples (either of these will work):

```
bash$ EDITOR=nano visudo   # call visudo with EDITOR=nano
```

or

```
bash$ export EDITOR=nano
bash$ visudo
```

Yum: Yellowdog Updater Modified

- http://teaching.idallen.com/cst8207/13w/notes/810_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
  - yum list installed
  - yum list available
  - yum list # combination of two above
  - yum search 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

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

watching ntpd work

- start ntpd in 10 seconds
- meanwhile, print the date every second
- You COULD do this if you wanted to see what effect ntpd has on the date

```
bash# (sleep 10; service ntpd start) &
bash# while true; do
> date
> sleep 1
done
```

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 a directory hierarchy

- sometimes you'll see this outdated idiom

bash$ tar cf - adir | (cd /some/dir; tar xf -)

- that's a reliable way to copy adir and everything below it to /some/dir
- a file name of "-" means stdin if we're extracting, x, or stdout if we're creating, c.
- the parentheses mean run in a subshell
- the cd /some/dir changes the dir of that subshell, and the tar xf - extracts the archive read from the stdin
- rsync is the modern way to do this

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/cst8207/13w/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

scp (cont'd)

- Whatever 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

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 dir 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