RHCE
BOOT CAMP

System Administration

INSTALLATION

- Installing RHEL 6 is a straightforward process when performed interactively. I expect every single person in here can install RHEL 6 from media.

- Unattended install using a Kickstart file is another matter entirely, though.

KICKSTART FILES

- Fortunately, Kickstart files are *extremely simple* to understand and create.
- A Kickstart file is a flat text file which answers all of the installation questions automatically. Therefore, logically, it contains details on:
  - Partitioning and filesystems
  - Software packages
  - Users, Groups, Passwords
  - Features, networking and more

KICKSTART FILES

- There are three primary means of creating a Kickstart file:
  - From scratch
  - From an existing Kickstart file (perhaps from a recent install?)
  - Using `system-config-kickstart`

LAB

1. Examine `/root/anaconda-ks.cfg`

2. Install and run `system-config-kickstart` and create a simple kickstart file to install a basic desktop RHEL 6 machine.

NETWORK CONFIGURATION

- There are two main approaches to configuring a machine for network access:
  - Static configuration
  - Dynamic configuration
- Static configuration uses set parameters for the configuration, which is known by the machine and the network and never changes. Generally used with servers.
- Dynamic configuration configures network machines on the fly, where a service on the network provides all configuration parameters to a machine when it joins the network. Generally used with workstations.

DYNAMIC CONFIGURATION

- Dynamic configuration is the easiest to use.
- The machine just needs to set up it's interfaces with the DHCP protocol.
- DHCP: Dynamic Host Configuration Protocol.
- A lease is obtained from the DHCP server, providing all network configuration details for the client. The lease expires after some amount of time and is renewed by the client to maintain network access.

STATIC CONFIGURATION

- Static configuration requires four configuration parameters in order to allow full network functionality:
  - IP Address
  - Netmask
  - Default Gateway or Router
  - DNS Server(s)

DNS?

- Domain Name Service: This is the glue between network names and IP addresses.

- Remember: Humans like names, computers like numbers. DNS is a service like so many others, mapping names to numbers and numbers to names. Mostly a convenience.

- Also provides for email functionality, geographic load balancing and limited service failover capabilities.

STATIC CONFIGURATION

- The first two components of static configuration are IP address and netmask.
- These provide LAN-level access
- To view current address:
  - `ip addr list`

GATEWAYS

- The third configuration parameter is the default gateway.
- Provides access to *inter-networking*, or moving from just the local LAN to other LAN's
- To see the current routing entries:
  - `ip route show`

DNS SERVERS

- Final piece of configuration information.
- List of one or more IP addresses which provide the DNS service, allowing name to IP address mapping
- To view current nameservers, see:
  - `/etc/resolv.conf`
- Also consider `/etc/nsswitch.conf`

STATIC CONFIGURATION

- Once all four pieces of information are configured on the system, full network service will be available.
- To test local connectivity, try pinging the gateway
- To test inter-networking connectivity, try pinging `8.8.8.8` or some other external IP address.
- To test name resolution, try pinging `google.com` or another public DNS name.

CHANGING NETWORKING

- To change the IP address, hostname, netmask and gateway, you have to edit two configuration files:
  - `/etc/sysconfig/network-scripts/ifcfg-em1`
  - `/etc/sysconfig/network`

/ETC/SYSCONFIG/NETWORK

```
NETWORKING={yes|no}

HOSTNAME=<fqdn>

NISDOMAIN=<nis domain name>
```

IFCFG-* FILES

- To configure a device to use dhcp, the ifcfg file should contain the following:

```
DEVICE=em1

BOOTPROTO=dhcp

ONBOOT=yes
```

IFCFG-* FILES

- To configure a device with static settings, the ifcfg file should contain the following:

```
DEVICE=em1

BOOTPROTO=none

IPADDR=<ip>

NETMASK=<netmask> ( or PREFIX=<net bits> )

ONBOOT=yes

GATEWAY=<gateway ip>

DNS1=<nameserver ip>

DOMAIN=<search domain>
```

NETWORK MANAGER

- In RHEL 6, Network interfaces are now handled via Network Manager. Some notable commands/tools:
  - `nmcli` - simple CLI to Network Manager
  - `nm-connection-editor` - excellent GUI tool for managing all network connections.
- On the test, you should decide if you are going to use Network Manager or not, and if so, only use NM and don't edit the config files by hand. Otherwise, disable NM and edit the files by hand. Your choice!

NAT CONFIGURATION

- NAT Configuration, `eth0` outside, `eth1` inside:

```
sysctl -w net.ipv4.ip_forward=1 >> /etc/sysctl.conf

iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE

iptables -A FORWARD -i eth0 -o eth1 -m state --state
RELATED,ESTABLISHED -j ACCEPT

iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT

service iptables save
```

STATIC ROUTES

- Static routes are configured via:
  - `/etc/sysconfig/network-scripts/route-eth0`
    - `192.168.0.0/24 via 192.168.0.1 dev eth0`
  - `/etc/sysconfig/network-scripts/route-eth1`
    - `10.10.10.0/24 via 10.10.10.1 dev eth1`

LAB

1. Determine your current network settings (which were assigned by DHCP) and change your machine to a static network configuration using these settings.

2. When you are satisfied with your configuration, restart the network service to put your changes into effect.

3. Test your connectivity to `server1` to make sure you are still online.

4. Refer back to DHCP settings if necessary to correct any mistakes in your static configuration.

5. Once complete, switch everything back to DHCP.

CRON

- `crond` is the cron daemon. Cron provides for the ability to execute commands on a regular basis.

- Generally used to run hourly, daily and weekly type system maintenance scripts.

- Also useful to run reports, cleanup jobs and much, much more.

SYSTEM CRONS

- `/etc/crontab` and `/etc/cron.d/*` define the system cron jobs.
- `/etc/anacrontab` defines system cron jobs that are run even if the machine was not running when the job normally executes.
- Many distributions use the `run-parts` script to execute all scripts found in `/etc/cron.hourly`, `/etc/cron.daily`, etc on the appropriate schedule.
  - `/etc/anacrontab` defines the times for each schedule: daily, weekly, monthly
  - Due to limitations in `anacrontab`, the hourly scripts are configured to run via `/etc/cron.d/0hourly`

USING CRON

- Cron is controlled through crontab files.
- There are system-wide crons as discussed previously.
- Every user has their own crontab, accessible through the `crontab` command

CRONTAB

- `crontab`: View, edit or remove crontabs
  - The `-l` option prints the crontab. The `-e` option opens the crontab for editing. The `-r` option removes the crontab.
- Root can work with the crontab for any user by specifying the username on the command line:
  - `crontab -e -u bob`

CRONTAB SYNTAX

- There are two main components to a crontab entry:
  - The timespec specifies when the command should be run
  - The command is what gets executed every time the timespec is matched

CRONTAB TIMESPECS

- The timespec is broken down into 5 fields, separated by spaces:
  - `minute hour day-of-month month day-of-week`
- Each field can contain a number, a range of numbers, a comma-separated list of numbers, an asterisk or a number slash division rate
- Mostly self-explanatory - some examples will help...

TIMESPEC EXAMPLES

- `0 23 * * *` *11pm every day*
- `30 * * * 1-5` *30 minutes after every hour, M-F*
- `0 7 1 * *` *7am, first of every month*
- `* * * * *` *Every single minute*
- `0,10,20,30,40,50 * * * *` *Every 10 minutes*
- `*/5 8-17 * * 1-5` *Every 5 minutes, 8am-5pm, M-F*

EXAMPLE CRONTAB

```
01 4 * * * /usr/local/bin/restart-webserver
00 8 1 * * /usr/bin/mail-report boss@mycompany.com
*/5 * * * * /monitor/bin/check-site -e admin@mycompany.com -o /var/log/check.log
```

- There are various additional options and features available to the cron system. Check the man pages for reference:
  - `cron`, `crontab` (sections 1 and 5)

LAB

1. Create a cronjob for the user root that checks the amount of available space on the system every Friday at 12:34pm.

2. Create a cronjob as a regular user that lists the contents of `/tmp` at 3:54am on Sunday, January 2. Hint: not possible with just cron syntax - you will have to do a tiny amount of scripting to complete this one.

LOGS

- One of the easiest places to find the cause of a problem is in the log files.

- Log files store informational messages from software. The types of messages include debug information, status information, warnings, errors and more.

- Some applications manage their own log files. Others use the system-wide logging package...

SYSLOG

- `rsyslog` - The system logger. A framework consisting of a library, a daemon, a configuration file and logs.
- Any application can use the library and log messages through `rsyslog` with simple function calls.
- Log messages consist of 3 parts:
  - Facility
  - Level
  - Message

SYSLOG

- The facility describes what part of the operating system generated the message, and is selected by the software:
  - `auth, authpriv, cron, daemon, ftp, kern, lpr, mail, news, security, syslog, user, uucp, local0-local7`
- The level represents the importance of the message, and is also chosen by the software:
  - `emergency, alert, critical, error, warning, notice, info, debug`

/ETC/RSYSLOG.CONF

- `/etc/rsyslog.conf` defines where all of the log messages should go. Destinations include files, screens of logged in users, console, other syslog servers. Additional configuration is available as well.

- Basic rule format:

  - `facility.level  destination`

- Examples:

  - `*.err  /dev/console`

  - `mail.*  /var/log/maillog`

  - `*.info;mail.none;authpriv.none  /var/log/messages`

/VAR/LOG

- `maillog`: messages from the email subsystem
- `secure`: authentication and security messages
- `cron`: cron messages
- `boot.log`: boot messages
- `messages`: catch-all
- `dmesg` : hardware and kernel events generated before syslogd

REMOTE LOGGING

- Setting up remote logging with rsyslog is trivial:
  - Make sure a hostname is set up on each machine
  - Make sure server firewall has holes for port 514 udp/tcp

REMOTE LOGGING SERVER

- On the server, add to `rsyslog.conf`:
  - `$ModLoad imudp.so`
  - `$UDPServerRun 514`
  - `$ModLoad imtcp.so`
  - `$InputTCPServerRun 514`
- Restart `rsyslogd`

REMOTE LOGGING CLIENT

- On the client, add to `rsyslog.conf`:
  - `*.*  @loghost.fqdn   # for udp`
  - `*.*  @@loghost.fqdn  # for tcp`
- Restart `rsyslogd`
- Consider using the Action Queue parameters to improve reliability. See bottom of `rsyslog.conf` for example.

LOGS

- As mentioned earlier, not all software uses the syslog framework to handle it's logging. Quite a bit of software manages it's own logs.

- This can make it difficult to track down all of the log locations on an unfamiliar system. The best way to handle this is to start from the init scripts...

LOCATING APPLICATION LOGS

- To track down the log file location for an application, you need to find it's configuration file so you can see where the logs are being written.

- Of course, finding the configuration file might be just as difficult, so it's best to start at the source.

- `init` starts all of the system services, and so there is an init script somewhere that is starting up the application in question.

- The init script almost always references the configuration file

LOCATING APPLICATION LOGS

- Now that the configuration file location is known, it only takes a few moments to scan through it and find out where logs are being written.

- As for the format of the log file, that's completely dependent on the application. Some will be similar to syslog, others, like Apache or Qmail, will be completely foreign looking.

- Fortunately, a little common sense and judicious application of Google Ointment will get the information you seek.

MAINTAINING LOGS

- `/etc/logrotate.conf`
  - This is the main configuration file for logrotate.
- `/etc/logrotate.d/`
  - EVERYTHING in this directory will be parsed as if it is a logrotate configuration file. Usually, applications such as Apache and Sendmail will have configuration files in this directory to control how their logs will be rotated.
- `logrotate -vf /etc/logrotate.conf`
  - Can be run as root at any time to force log rotation and check for errors.

TROUBLESHOOTING

- There will be some basic troubleshooting objectives on the exam, mostly to test basic knowledge of how permissions should work, SELinux and locating error messages in log files.

- Mentioned here are a few useful tools to remember

TOP

- `top`: Self-updating tool displays combination summary at top, followed by ordered list of processes. Fully customizable.
  - The summary includes uptime information, memory breakdowns, CPU utilization and process state summaries
  - The process display can be customized and sorted to suit need

```
top - 16:39:32 up 682 days, 10:41,  2 users,  load average: 0.01, 0.00, 0.00
Tasks: 118 total,   1 running, 116 sleeping,   1 stopped,   0 zombie
Cpu(s):  0.1%us,  0.0%sy,  0.0%ni, 99.8%id,  0.0%wa,  0.0%hi,  0.0%si,  0.1%st
Mem:    262316k total,   258024k used,     4292k free,     7380k buffers
Swap:   524280k total,    74564k used,   449716k free,    67808k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
    1 root      15   0 10316  648  592 S    0  0.2   0:06.24 init
    2 root      RT   0     0    0    0 S    0  0.0   0:04.88 migration/0
    3 root      34  19     0    0    0 S    0  0.0   0:00.19 ksoftirqd/0
```

DF

- `df`: lists filesystem utilization
  - Breaks down size and use information for each mounted filesystem
  - -h is useful option to display in “human-friendly” format

```
[root@dev1 ~]# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda1             9.4G  7.2G  1.8G  81% /
none                  129M     0  129M   0% /dev/shm
[root@dev1 ~]#
```

LDD, LDCONFIG

- `ldd`: List library dependencies
- `ldconfig`: Update library location database
- `/etc/ld.so.conf` and `/etc/ld.so.conf.d/*.conf` for list of pathnames to search for libraries, creates database for dynamic linker

```
[root@dev1 ~]# ldd /bin/bash
    libtermcap.so.2 => /lib64/libtermcap.so.2 (0x00002ac044572000)
    libdl.so.2 => /lib64/libdl.so.2 (0x00002ac044775000)
    libc.so.6 => /lib64/libc.so.6 (0x00002ac044979000)
    /lib64/ld-linux-x86-64.so.2 (0x00002ac044357000)
[root@dev1 ~]# cat /etc/ld.so.conf.d/mysql-x86_64.conf
/usr/lib64/mysql
[root@dev1 ~]# ldconfig
[root@dev1 ~]#
```

NICE LEVEL

- The nice level represents one influence on the calculations the kernel uses when assigning priorities.
- Originally designed and named to allow users to be “nice” to other users of the system by assigning a higher nice value to an intensive process, which in turn lowers it’s priority.
- Ranges from -20 to 19. Default nice level is 0.
- Only root can assign negative nice values.
- See `nice` and `renice` commands

LAB

1. Take a few minutes to browse through the various logs in `/var/log`. Familiarize yourself with the kinds of information available.

2. Browse the man page for `rsyslog.conf`

3. Find where the audit service keeps its log and add a corresponding new entry to your logrotate configuration. Force a rotation to see everything work.

4. Remove the audit logrotate configuration and restart the auditd service.

5. Locate the PIDs of the highest memory and highest CPU utilization processes. Play with their nice levels.

6. Work with a neighbor to set up remote logging from your station to theirs, and theirs to yours. Verify using `logger`. Careful of log loops!

SHELL SCRIPTING

- Shell scripting involves placing a series of shell commands in a file for later re-use.
    - Simple shell scripts simply run command after command, as if the user typed them in at the command line
    - More complex shell scripts actually make decisions about what commands need to be run, and might even repeat certain sequences to accomplish some task
- Scripts start executing at the top and stop when there are no more commands to execute or when `exit` is called.

EXAMPLE SHELL SCRIPT

- Here is an example of a very simple shell script:

```
echo “Hello, what is your name?”
read NAME
echo “Hello $NAME, it’s nice to meet you!”
echo -n “The current time is: “
date
```

- Using the `echo` command, this script asks a question.
- The `read` command accepts input from the user and stores it in the environment variable `NAME`
- The script finishes up with a couple more echo statements, greeting the user and announcing today’s date

SHELL SCRIPTING

- If we put the example in a file called `myscript`, we can execute the script as:
  - `bash myscript`
- `bash` will open `myscript` and execute each line as if the user had typed it in manually.

```
[root@localhost ~]# bash myscript
Hello, what is your name?
Linus
Hello Linus, it's nice to meet you!
The current time is: Sun Nov 29 09:39:33 CST 2009
[root@localhost ~]#
```

INTERPRETERS

- In the previous example, we put five commands in a regular file and fed the filename to `bash` on the command line, which in turn executed the commands.

- Running in this way, `bash` operated as an interpreter. Reading each line of the file, `bash` would interpret the words and perform some action.

- There are many interpreted languages available for scripting, including all shells, `python`, `ruby`, `perl`, etc.

EXECUTING SCRIPTS

- To run a script, feed the file to the appropriate interpreter:
  - `bash mybashscript`
  - `perl myperlscript`
- This works fine, but sometimes it's more user-friendly to allow the script to be run directly, removing the need for an external call to the interpreter...
  - `./mybashscript`
  - `myperlscript`

SHEBANG

- This is accomplished with the shebang (#!). Also known as a hash bang, pound bang or hashpling.

- When the kernel is asked to execute a file, it must either be machine code, or a file that starts with the shebang sequence. If the first two characters of the file are a hash mark and an exclamation mark, the rest of the line is expected to be an absolute pathname for an interpreter, which will then be invoked to “run” the file as a script.

SHEBANG

- So, add an appropriate shebang to the example:

```
#!/bin/bash
echo “Hello, what is your name?”
read NAME
echo “Hello $NAME, it’s nice to meet you!”
echo -n “The current time is: “
date
```

- Then add execute permissions and the script can be run directly:

```
[root@localhost ~]# chmod 755 myscript
[root@localhost ~]# ./myscript
Hello, what is your name?
Linus
Hello Linus, it’s nice to meet you!
The current time is: Sun Nov 29 09:39:33 CST 2009
[root@localhost ~]#
```

DECISIONS

- More advanced problems require the script to make decisions. There are two basic ways to make decisions with shell scripts:
  - `if` statements
  - `case` statements

TEST COMMAND

- Before we continue talking about decisions, we need to talk about the `test` command. This command actually performs the comparisons necessary to ask a question, such as:
  - `"string1" = "string2"` *Returns true if string1 is identical to string2*
  - `VAR -le 45` *Returns true if VAR is numerically less than or equal to 45*
- See the man page for `test` for additional details

IF STATEMENTS

- Basic syntax:

```
if list;

  then list;

  [ elif list; then list; ]

  ...

  [ else list; ]

fi
```

IF EXAMPLE

```
#!/bin/bash
echo “Hello, what is your name?”
read NAME
if [ “$NAME” = “Linus” ]
then
  echo “Greetings, Creator!”
elif [ “$NAME” = “Bill” ]
then
  echo “Take your M$ elsewhere!”
  exit
else
  echo “Hello $NAME, it’s nice to meet you!”
fi
echo -n “The current time is: “
date
```

- This script will now base it’s response based on what name the user provides

CASE STATEMENTS

- Basic syntax:

```
case word in
  pattern [| pattern] ) list;;
  ...
esac
```

CASE EXAMPLE

```
#!/bin/bash
echo “Hello, what is your name?”
read NAME
case $NAME in
  “Linus” )
    echo “Greetings, Creator!”
    ;;
  “Bill” )
    echo “Take your M$ elsewhere!”
    exit
    ;;
  * )
    echo “Hello $NAME, it’s nice to meet you!”
esac
echo -n “The current time is: “
date
```

- This script also bases it’s response based on what name the user provides, but does so using a case statement instead of a large if statement

LOOPING

- Sometimes a certain sequence of commands need to be run repeatedly, either for a set number of times or while some condition is true. This is accomplished with:
  - `while` loops
  - `for` loops

WHILE LOOPS

- Basic syntax:

```
while list;

  do list;

done
```

WHILE EXAMPLE

```
#!/bin/bash
echo “Hello, what is your name?”
read NAME
while [ “$NAME” != “Linus” ]
do
  echo “I don’t know that person, what is your name?”
  read NAME
done
echo “Greetings, Creator!”
echo -n “The current time is: “
date
```

- This script will loop until the name typed is “Linus”

FOR LOOPS

- Basic syntax:

```
for (( expr1 ; expr2 ; expr3 ))

  do list;

done
```

FOR EXAMPLE

```
#!/bin/bash
echo “Hello, what is your name?”
read NAME
for (( I=0 ; I<3 ; I++ ))
do
  echo “Hello $NAME!!”
done
echo -n “The current time is: “
date
```

- This excitable script repeats your name 3 times before giving you the date and time

SCRIPTING

- There is of course quite a bit more to shell scripting than can be covered in this course. There are a few more structures you can use for looping, and dozens of special metacharacters for achieving all kinds of results.

- With this introduction, though, you should be able to read through light shell scripts and have a handle on what's going on, as well as be able to write simple ones on your own.

EXERCISES

- Write a simple shell script that prints out the message “Hello world.” Make the script executable and verify it works correctly by running it as “./myscript”

- Browse through the man page on ‘bash’, focusing in a bit on the various scripting elements.

- Devise a program which uses one or more **if** statements, changing behavior based on user input or command line arguments. Output a warning to stderr if there are no arguments passed to your script on the command line.

```
slideshow.end();
```