Red Hat Certified System Administrator (EX200) Notes

Table of Contents

Preface

This document is a collection of notes taken by me, Kevin Clark, over the summer months of 2017 using CentOS 7.3. The test is on RHEL7. This document was created August 2017, and finished in November 2017. Any revision history can be found at the end. These notes cover topics I think are relevant to the exam per the EX200 exam objectives. I took the EX200 test on June 29th 2018 and passed with a 250/300 (210 is the lowest passing score). Some material was sourced from the RedHat student handbook borrowed from my teacher John Domagall, and the rest was found online. Primarily I used CertDepot.

Document formatting:

#Comment

File contents

Section:

Topic

$variable

Example:

/file/name

/directory/path/

/symlink/path
servicename
TL;DR:
> Bullet Point
[This|That] – This or That (not both)

Prerequisites:
These notes expect that the reader has some experience with the topics covered, and a solid understanding of the following:
> Variables, how to set, unset, and recall.
> Basic rwx permissions, SGUID, SUID, and sticky bit.
> Linux file and directory structure, hidden files.
> File, stat, ls, cp, mv, rm commands.
> Reading and writing files with a CLI text editor (I recommend vim).
> Grep, shell pattern matching/wildcards (sometimes confused with regex), awk, cut, pipes.
> Links, hard and symbolic (soft).
> Partitioning concepts and storage size knowledge.
> Man and --help page usage including searching.
> Basic service management.
> Find command, and the name and type options at a minimum.
> Basic understanding of runlevels.
> Networking concepts: IP address, netmask, gateway, DNS, DHCP, ports and firewall basics.
> Root user privileges and the difference between root and a normal user.
> Installing executable software (commands) and daemons using yum.
> Creating, modifying, and deleting user accounts.
> Client Server model for network services.
> SSH usage.
> Command substitution with backticks (`subcommand`) or parentheses ($(subcommand)).

Grub2
TL;DR: Grub2 is the default Linux bootloader. It is executed before the kernel and controls which kernel is loaded. Grub can also be used to acquire root on a Linux machine without the password assuming the user has physical console access.

Files:
> /boot/grub2/grub.cfg #Master configuration file. Do not edit by hand.
> /etc/grub2.cfg → /boot/grub2/grub.cfg #Symlink to previous file.
> /etc/default/grub #This file can be edited to make changes to the grub configuration.
> /etc/sysconfig/grub → /etc/default/grub
> /etc/grub.d/ #Contains grub internal files. Do not mess with.

Commands:
> grep ^menuentry /boot/grub2/grub.cfg #Shows all entries of kernel boot options.
> grub2-set-default *$number* #Where 0 is the first, set the default kernel boot.

> grub2-mkconfig -o /boot/grub2/grub.cfg #Applies any changes made in /etc/default/grub for the next boot. Without this, grub file changes will not be applied. Note that grub2-set-default will still be applied without this.

Yum/Package Management

TL;DR: Yum is the default Redhat package manager. It can be used to download software from internet or local repositories. It tracks dependencies and eliminates a lot of headache of manually installing software.

Files:

> /etc/yum.conf #Yum configuration file.
> /var/log/yum.log #Logging file for Yum. Can be changed in the yum configuration file.
> /etc/yum.repos.d/*.repo #Individual repo files. You will be required to set up a repo manually by creating a repo file.

Notes:

Example: Adding a new repo for a repo file in /etc/yum.repos.d/:

```
[repository_name_id]  #Cannot contain spaces.
name=" Repo Name"  #Long form repo name.
baseurl="http://www.url.com"  #Can also be https://, ftp://, or file:///.
enabled=1  #Boolean. 0 for disabled, 1 for enabled.
gpgcheck=1  #Boolean. 0 for skip gpgchecking, 1 to enable gpg.
gpgfile="file:///path/to/gpg/file.txt"  #gpg file.  Can also be http, https, ftp.
```

> When creating a local repo (using file:///path/localrepo/) make sure to put the repository above the packages and repodata folders.
> The /dev/sr0 device is the default CD location. This is useful for local repos or recovery booting.
> One broken repo can cause all of yum to fail. Turn repos on and off with "enabled=1" or "enabled=0"
> The --nogpgcheck command-line option and gpgcheck=0 in a repo file will disable gpgchecking. If you get errors installing something with yum, try adding –nogpgcheck to your command.

Rpm Commands:

> rpm --import /path/to/gpg-key #Imports a gpgkey for a repo. You'll get yum gpg errors otherwise.
> rpm -qa #Lists all installed packages.
> rpm -qf *$file* #Shows the package where the file originated from.
> rpm -ql *$package* #Shows all the files that make up the specified package.

Yum Subcommands:

> [local]install [*$package*|*$rpmfile*] #Installs the package from the repo or rpm file.
> update [*$package*|*$rpmfile*] #Updates the package from the repo or rpm file.
> remove *$package* #Uninstalls and removes the package.
> info *$package* #Shows a long block of information about the package.
> list *$string* #Shows all packages that match the entire string.
> search *$string* #Shows all packages that contain the string
> deplist *$package* #Shows the dependencies that the specified package relies on.

> repolist [all|enabled|disabled] #Shows the current yum repos.
> group[list|install|remove|info] #Commands that deal with package groups.
> grouplist hidden #For some reason, many package groups are hidden and don't show up with just a normal "yum grouplist". Use this to see all package groups.
> whatprovides *$file* #Shows what package needs to be installed to get the file. Useful if you know the name of a command but forgot the name of the package that installs the command.

Systemctl

TL;DR: Systemctl is the replacement for RHEL6's service and chkconfig commands and one of the ways to interact with the replacement init system, Systemd. Like RHEL6 service and chkconfig commands, it controls services persistently and for the current boot. It also is in charge of targets (like runlevels in RHEL6). Services, targets, and more are collectively known to Systemd as units.

Notes:
> rescue.target is similar to single user mode/runlevel 1.
> multi-user.target is like runlevel 3. CLI only.
> graphical.target is what it sounds like: graphical. Similar to runlevel 5.
> Shutdown, poweroff, and reboot are also targets. These are actually symlinks to systemctl commands.
> The .service suffix can be left off systemctl commands if desired. Systemctl is smart enough to recognize a service by its name.
> All systemctl subcommands can be viewed by using autocompletion: systemctl <tab><tab>

Systemctl Subcommands:
> start *$service*.service #Starts service immediately, but only for the current boot.
> stop *$service*.service #Stops service immediately, but only for the current boot.
> enable *$service*.service #Enables the service to start at boot time.
> disable *$service*.service #Disables the service from starting at boot time.
> reload *$service*.service #Re-reads the config files of a service without stopping the service. Sometimes doesn't work. For this reason, prefer restart over reload.
> restart *$service*.service #Stops and then restarts the specified service.
> status *$service*.service #Shows the status of the current service. Running, stopped, or failed.
> list-unit-files #Shows all units systemctl can recognize.
> cat *$unit* #Reads the systemctl file associated with the unit.
> daemon-reload #Re-read all systemctl unit files into systemctl after a manual unit file change.
> get-default #Shows what target systemctl will boot into.
> set-default *$target*.target #Changes the default target that systemctl will boot into.
> isolate *$target*.target #Changes the current target to the specified target for the current boot.

Networking

TL;DR: Networking on Linux is best done manually. On the test, you will be given an IP address, netmask, gateway, and DNS server address. All these need to be put into the right config files to allow Linux to network with systems around it.

Files:
> /etc/sysconfig/network-scripts/ifcfg-$*interface* #File contains the settings for the network interface. IP address, netmask, gateway, onboot, dhcp/static. Interface is usually looks like "enp#s#" or "eth#".
Example: /etc/sysconfig/network-scripts-ifcfg-enp0s3:

```
TYPE=Ethernet
BOOTPROTO=dhcp  #Can be set to static with IPADDR, GATEWAY and NETMASK.
NAME=enp0s3
DEVICE=enp0s3  #NAME and DEVICE should be equal to each other.
ONBOOT=yes  #Initializes network interface at boot-up.  Keep this as yes.
DNS1="8.8.8.8"
DNS2="8.8.4.4"
```

> /etc/resolv.conf #Holds DNS servers to use.
Example: /etc/resolv.conf:

```
nameserver 8.8.8.8
nameserver 8.8.4.4
```

> /etc/hosts #Allows local name resolution.

Commands:
> systemctl disable **NetworkManager** ; systemctl stop **NetworkManager** #NetworkManager sucks. Configure the network manually. In the words of John Domagall, "Something as critical as networking is best left to the professionals."
> systemctl [restart|reload] **network** #Reapplies settings changes in the network files.
> systemctl status **network** #Shows the status of networking.
> hostname #Shows current hostname
> hostnamectl set-hostname $*hostname* #Set's the hostname currently and for all future boots.
> ip a #Shows ip addresses and other details for all network interfaces.
> ip route #Shows the ip routing table.

Root Break-in

TL;DR: Root break-in is a method of resetting the root password when physical access to the console is given.

Method: rd.break #This is the preferred method. It is officially recommended by Redhat.
1. Boot up the machine and press "e" at the grub2 boot menu to edit the grub line.
2. Find the line near the bottom that starts with "linux"
3. Add "rd.break" to the end of the line (without the quotes).
4. Press Ctrl+X to execute the boot.
5. At the pound (#) prompt, execute the following: mount -o remount,rw /sysroot
6. chroot /sysroot
7. passwd #Change your password
8. touch /.autorelabel #Tells the system to re-label the SELinux contexts of the files you changed
9. exit #Exits the chroot.
10. reboot

11. Log in

Extended File ACL's

TL;DR: Allows more granular permissions on files for individual users and groups.

Notes:

> Remember that g+s permission (sguid) on a directory allows for "collaboration". This means that any file created inside the directory will have the same group ownership of the directory.

> -R option will recurse to all subfolders and files

Commands:

> getfacl *$filename* #Displays the current ACL settings.

> setfacl [-m|-x] *$permset $filename* #Adds (m) or removes (x) the permissions set as an entry to the ACL.

Examples:

setfacl -m u:kevin:rwx faclfile #Adds the kevin user to the ACL with rwx.

setfacl -m g:sysadmins:r-x faclfile #Adds the sysadmins group to the ACL with r-x.

setfacl -x u:kevin faclfile #Removes the kevin user entry from the ACL.

Turning Off SELinux

TL;DR: SELinux is a pain in the ass; here's how to turn it off. Remember that enforcing is on and permissive is off.

Notes:

> Turn off SELinux by changing the file /etc/selinux/config. Change SELINUX=enforcing to SELINUX=permissive. This requires a restart to take effect.

Commands:

> getenforce #Shows current SELinux enforcement.

> setenforce [permissive|enforcing] #Sets SELinux enforcement for the current boot.

Group Management

TL;DR: Group creation, modification, and deletion, file ownerships, and primary and secondary groups.

Notes:

> A user's primary group is applied to any files or directories they create as group ownership.

> Secondary groups are just used for the users in the group to have access to files and directories with that group ownership.

Commands:

> groups *$username* #Shows group membership of the specified user.

> usermod -g *$group $username* #Changes the user's primary group to the group specified. A user can only have one primary group at a time.

> usermod -G $groups $username #Sets all groups listed as secondaries. The primary group will not be effected. All previous secondary groups will be removed. Use -aG instead to append to the already existing secondary groups.
> chown *$user*:*$group* *$filename* #Changes the user and group ownership on a file. Must be root to do this.
> [groupadd|groupdel] *$group* #Creates or deletes a specified group.
> chage #Command used to manage account and password expiry. See chage --help for info.

File:
> /etc/group #Holds the group entries in a similar way to /etc/passwd.

SSH, SCP, and SSH Keys

TL;DR: SSH autologin keys allow password-less entry to hosts that have it set up. SCP allows secure file transfer from one host to another. Both of these use port 22 (by default) and the server must have sshd turned on to use these utilities.

Notes:
> Root login is disabled in SSH by default. To enable it, remove the # in the "#PermitRootLogin yes" line in /etc/ssh/sshd_config file. Then restart **sshd**. Root logon should never be allowed for systems exposed to the internet as it's a huge security concern.
> SCP needs a user, host, and file for both source and destination implicitly or explicitly. This means a total of 6 parameters are needed. To make things shorter, some can be omitted. When referencing a local file, user and host can be omitted.

Files:
> /etc/ssh/sshd_config #Holds sshd service settings. Modify with a text editor and restart sshd to see affects.
> /etc/ssh/ssh_config #Holds the client SSH settings. Modify to change settings.
> ~/.ssh/authorized_keys #The location to put public keys on a remote server. This file needs 600 permissions otherwise sshd will refuse to start. SCP and SSH can be used to transport and set up this file on the remote host.
> ~/.ssh/id_rsa #Private key that ssh-keygen generates. Do not touch this file after it is generated.
> ~/.ssh/id_rsa.pub #Public key that ssh-keygen generates. To set up SSH autologin keys, the contents of this file need to be put in the remote host's authorized_keys file.

Commands:
> ssh *$username*@*$hostname* #Attempts to log in to the remote server with the specified username and a password that the user supplies.
> ssh-keygen #Generates a public private key pair and puts them into ~/.ssh/id_rsa.pub and ~/.ssh/id_rsa respectively.
> scp *$user*@*$sourcehost*:*$filepath* *$user*@*$destinationhost*:*$filepath* #Transfers a file from the source host to the destination host using each user's credentials.
Example: scp ~/.ssh/id_rsa.pub root@server1:/root/.ssh/id_rsa.pub.putinauthorized_keys #Pushes the public key from the local machine to server1 to a file called id_rsa.pub.putinauthorized_keys.

Storage Discovery

TL;DR: This section shows how to view information about traditional partitions, LVM, filesystem types, labels, UUIDs, swap, memory, and disk usage, and processes.

Notes:

> The first letter of a device name, S, H, or V, means the device is SCSI, IDE, or virtualization aware respectively.
> The last letter of a device name means the number of that device. A is 1, B is 2, etc.
> The ending number indicates what partition that device is.

Example: /dev/sdb3 – This is the third partition of the second SCSI device.

Commands:

```
> lsblk -f  #Shows filesystem types, device label, and UUID.
> lsblk  #Shows device tree, size and type.
> [pvs|vgs|lvs]  #Shows brief information about physical volumes, volume groups, and logical volumes.
> [pvdisplay|vgdisplay|lvdisplay]  #Shows detailed information about physical volumes, volume groups, and logical volumes.
> fdisk -l  #Shows some useful disk information like partitions for devices and whether they are bootable or not.
> free -h  #Shows memory and swap usage.
> mount  #Shows mounted file systems and if they are read/write or read only.  Grep this one as there is a lot of garbage information that comes through.
> top  #Shows top processes by resource usage in real time.
> vmstat 1 10  #Shows resource usage over 10 seconds.  Ignore the first line.  It's a system summary.
> df -h  #Shows disk usage for mounted devices.
> df -h $path  #Shows device of specified path.
> swapon -s  #Shows swap devices.
```

Files:

```
> /dev/mapper/  #Holds all LV's
> /dev/$volumegroup/  #Holds just that volumegroup's LV's
> /etc/fstab  #Contains instructions on how to mount devices on boot (or on mount -a).  Uses spaces or tabs as delimeters between fields.
```

Example: /etc/fstab fields

```
#Field 1: Storage Device Name.  Device name, nfs mount, device label, or device UUID.
#Field 1 examples: /dev/sda1, UUID=123abcDEF, LABEL=device1, nfs_host:/path/to/files.
#Field 2: Mount Point.  Empty directory to mount onto.  For swap, enter "none" or "swap".
#Field 3: Filesystem Type.  Examples are ntfs, swap, nfs, ext4, xfs.
#Field 4: Mount Options.  Usually stick to "defaults" unless other options are needed.
#Field 5: Dump.  Set to 0.
#Field 6: Fsck Order.  Set to 0 to disable or 2 for non-priority fsck.
```

Mount Management, Filesystems, Swap, and Labels

TL;DR: To mount a partition/logical volume, you must first create the partition or locgical volume (LV) out of the device, format it with a filesystem, then mount it. To keep this persistent, you must also create an entry in /etc/fstab.

Notes:
> The two most common filesystems are xfs and ext4. Xfs is the RHEL7 default filesystem. RHEL6 used ext4. Both are standard and commonly used.
> Note that ext4 can both be shrunk and extended, while xfs can only be extended, not shrunk.
> Formatting a new filesystem onto a disk/partition/logical volume will destroy all current data on it.
> Make sure the directory being mounted on is empty! Otherwise overmounting will occur.

Commands:
> mount /dev/*$devicename* *$directorypath* #Mounts the device on the directory specified. If the mount is already in /etc/fstab, the device name or directory can be omitted.
> mount -o remount,[rw|ro] *$directorypath* #Mounts the filesystem as read/write or read/only. Useful for root break-ins.
> mount -a #Attempts to remount all (non swap) mounts listed in /etc/fstab as long as they have the auto option, which is in the defaults. Always test your changes to /etc/fstab with this command.
> swapon -a #Attempts to remount all swap mounts listed in /etc/fstab as long as they do not have the noauto option.
> umount [/dev/*$devicename*|*$directorypath*] #Unmounts the Filesystem of the directory or device.
> mkfs.*$fstype* /dev/*$devicename* #Formats the filesystem onto the device. You can type mkfs and then tab-tab to see all possible file system types.
> mkswap /dev/*$devicename* #Formats the device with swapfs, the swap file system. This needs to be done before a device can be used as a swap space. Can also be used to format files as swap instead of devices.
> swap[on|off] /dev/*$devicename* #mounts or unmounts a swapfs formatted device to or from the Linux swap space/paging pool.
> swaplabel -L *$label* /dev/*$devicename* #Changes the label of a swap device.
> xfs_admin [-U|-L] /dev/*$xfsdevice* #Changes the UUID or label of the specified xfs device.
> e2label /dev/*$devicename* *$label* #Changes the label of an ext* device
> xfs_growfs /dev/*$devicename* #Used to grow an xfs filesystem after an lvextend on the device.
> resize2fs /dev/*$devicename* #Used to grow an ext* filesystem after an lvextend on the device.
> uuidgen #Simple command to generate a UUID in case a new one is needed.

Traditional Partitioning

TL;DR: Traditional partitioning uses fdisk, a menu based command. Each disk can have up to 4 partitions. Either 4 primary (real), or 3 primary and one extended (logical). The extended partition allows more logical partitions past the 4th. Fdisk uses a starting and ending sector number for partitions. **Example:** First sector: 2048, Last sector: 2500000.

Notes:
> For creating an extended partition, use partition number 4 and give all remaining space. Extended partitions (partitions 5 and up) will take from the space allocated to partition 4.
> Make sure to format the device or partition with a filesystem after creation otherwise it is unusable.

> During the storage amount selection in fdisk, the user is prompted for a range of bytes. Instead, a user can enter the first value as default, then write + before an amount of storage space to add that much. **Example:** First sector: 2048, Last sector: +1G #Adds 1 Gb.

Commands:
> fdisk /dev/*$devicename* #Enters the user into a disk configuration menu.
> fdisk -l /dev/*$devicename* #Shows the partitions of the specified device.
> partprobe #Updates the operating system's awareness of new partitions and devices.

Fdisk letters:
> m #Displays fdisk help.
> n #Creates a new partition – primary or extended.
> d #Deletes a partition.
> q #Quits without applying previous changes made.
> w #Writes previous changes made to the disk and exits

Logical Volume Manager (LVM2)

TL;DR: LVM2 deals with 3 different object types: Physical Volumes (PVs), Volume Groups (VGs) and Logical Volumes (LVs). PVs are physcial disks dedicated to LVM. These can be grouped together into a storage space pool called a VG. Space in a VG can be carved off to create an LV, which to the GNU/Linux operating system, is viewed as a storage device.

Notes:
> PVs need to be dedicated to LVM before use. /dev/sda or /dev/sdb1 would be examples of PVs, but only if they were made into LVM PVs first.
> An LV can be located at /dev/mapper/*$vgname*-*$lvname* or in /dev/*$vgname*/*$lvname*. Both paths refer to the same LV, and both are common ways that Linux denotes LVs. They are interchangeable, so know both.
> Since no human can remember all of the LVM commands (there are many), type in [pv|vg|lv] at the command line and then press Tab-Tab to let Bash do the remembering for you.

Commands:
> pvcreate /dev/*$devicename* #Marks a physical disk device or partition as a PV able to be used by LVM.
> pvremove /dev/*$pvdevice(s)* #Removes LVM2 headers from device, and converts the device back into a normal, non-LVM, device.
> vgcreate *$vgname* /dev/*$pvdevice(s)* #Creates a VG with the specified VG name, and includes one or more PVs.
> vgextend *$vgname* /dev/*$pvdevice(s)* #Extends the specified VG with one or more PVs.
> vgreduce *$vgname* /dev/*$pvdevice(s)* #Does the opposite of above. Removes one or more PVs from a VG storage pool. This can be dangerous as removing space from a VG can destroy an LV, resulting in data loss.
> lvcreate -L *$size* -n *$lvname* *$vgname* #Creates an LV with specified name and size in Mb, taking storage space from the VG.

> lvremove /dev/*$vgname*/*$lvname* #Erases the LV and adds the newly freed storage back to the parent VG.
> lvextend -L +*$additionalsize* /dev/*$vgname*/*$lvname* #Extends the size of specified LV by adding the additional size provided. This space is taken from the parent VG. If used without the +, lvextend will extend the disk to the specified size.

KVM and Virsh

TL;DR: KVM is the default RHEL7 virtualization platform. Virsh is the management command for KVM virtual machines. Though large in scope, there are only a few commands needed for the RHCSA.

Notes:
> KVM only works on systems that support virtualization. Usually only works on physical machines. Must be 64 bit.
> Virt-manager and virt-viewer to create/manage and view VMs in a graphical environment, and virsh and virt-install to manage/use and install VMs in a non-graphical environment.

Commands:
> cat /proc/cpuinfo | grep -E 'svm|vmx' #If no output, KVM is not supported.
> yum groupinstall "Virtualization Host" #Installs KVM and required management commands.
> systemctl enable --now libvirtd #Starts the virtualization daemon now and persistently.
> virsh list --all #Shows all virtual machines: both running and powered off.
> virsh start *$vmname* #Starts up the VM.
> virsh reboot *$vmname* #Restarts the target VM.
> virsh shutdown *$vmname* #Shuts down the target VM.
> virsh console *$vmname* #Accesses the console of the target VM.
> virsh edit *$vmname* #Edits the xml settings sheet for the VM.
> virsh destroy *$vmname* #Pulls the plug on the VM.
> virsh undefine *$vmname* #Deletes the VM.
> virsh help *$category* #Help on a specific category of commands. For host commands, use "domain".

Files:
> /etc/libvirt/qemu/*$vmname*.xml #Xml file containing all VM settings. Do not edit manually as changes will be overwritten. Useful to view settings.

NFS

TL;DR: NFS (Network File System) is a common service that allows a server to share out specified files and directories to be accessed by remote clients.

Notes:
> NFS uses port 2049 by default.
> A root user (uid 0) trying to access files on a remote server will, by default, become the nfsnobody user. This user has very limited permissions and will not be able to access most files. This concept is called root squash.

> NFS uses user id numbers (uid) to identify users, not names. This means a client user account Kevin with uid 1000 and a server user account Allen with uid 1000 will be seen as the same user by NFS.
> In order for an NFS client to be able to read and write to a remote share, 4 conditions must be met. First, the user has to have the correct Unix (rwx) permissions as defined by the server. Second, the server filesystem must be mounted read-write. Third, the NFS share must be shared as read-write. Finally, the client must have the share mounted as read-write.
> An NFS server must have the nfs-utils package installed in order to start the **nfs** daemon.

Commands:
> showmount -e *$NFSserver* #Shows the available mounts being offered by the NFS server.
> exportfs -v #On the NFS host, shows NFS mounts being offered and the options of those mounts.
> exportfs -[u]a #Will export or unexport all NFS mounts listed in /etc/exports. Make sure to re-export after a change to /etc/exports.

Files:
> /etc/exports #Contains a server's outgoing exports.
Example: Simple /etc/exports file:
/NFSdata *(rw) #Shares the local /NFSdata directory out to all clients (*) with read-write permissions.
/abc host1(rw,sync,nohide) #Shares the /abc directory with only host1 with the options in parentheses.

> /etc/fstab #On a client, put NFS shares here to enable persistence across reboots. /etc/fstab fields were covered in the Storage Discovery section.
> /etc/nfsmount.conf #NFS configuration file. Includes options like version number, port, security version, timeout, etc.

Autofs/Automount

TL;DR: Autofs is a client extension for network file services. NFS is one such example. Autofs mounts network file shares on demand. This means mounts are not using hardware and network resources unless they are actively being used.

Notes:
> To use automount, make sure you have the nfs-utils and autofs packages installed.
> Directories configured for automounting will not be visible (via ls or other methods) until the directory itself is accessed. The easiest way to do this is to cd to the directory.
> A direct map (absolute pathing) uses '/-' in field 1 of /etc/auto.master with the full path in the child map file.
> An indirect map (relative pathing) splits up the full path into two parts; the first goes in /etc/auto.master, and the second goes into the map file.
> Restart **autofs** service after any changes to /etc/auto.* files.
> No (*) wildcarding in direct maps.
> Any wildcarding must be done at the last line of a map file.

Files:
> /etc/auto.master #Master file that holds pointers to the child map files.

> /etc/auto.*$name* #Child maps. The name can be whatever as long as it matches up with whatever is in /etc/auto.master. These files also need to be created manually.
> /etc/autofs.conf #Holds configuration for autofs.
> /etc/autofs_ldap_auth.conf #Configuration for using LDAP with autofs.

Example: /etc/auto.master:
```
/-        /etc/auto.direct  #Direct map
/home  /etc/auto.home  #Indirect map
```

/etc/auto.direct:
```
#Child map
/mnt/data1       host:/nfs/share
```

/etc/auto.home:
```
#Child map
kevin   host:/home/kevin
dave    host:/home/dave
chris   host:/home/chris
#All home directories can be automounted with the following syntax:
*         host:/home/&  #Where /home/* are the home directories.
```

Commands:
> yum install autofs nfs-utils #Installs the packages required for autofs.
> showmount -e *$server* #Shows the NFS shares that are available for client mounting.

Chrony

TL;DR: Chrony is a network time service that can act as a client or a server. Chrony is split into chronyc, the chrony client, and chronyd, the chrony daemon, or service. Chronyd is the replacement for ntpd.

Notes:
> Chronyd uses port 123 TCP and 323 UDP.
> Use this website to find ntp servers to draw from.
> The password must be entered in the chronyc prompt.

Example: chronyc
```
chronyc> password
Password: password1234
```

Files:
> /etc/chrony.conf #Chrony configuration file and server entry location. Put “allow” on a single line to allow the server to serve the ntp service to ntp clients, or “allow host” to allow only the specified host to get the ntp service. Remember to restart chronyd after every file change.

Example: /etc/chrony.conf time sources:
```
server red.example.com iburst
server yellow.example.com iburst
pool pool.ntp.org iburst  #iburst means “initial burst”.  This speeds up time sync when a server reboots.
```

> /etc/localtime #Symlink to a timezone file. By default, /usr/share/zoneinfo/America/New_York. Change symlink pointer to change the time zone.

Commands:
> timedatectl list-timezones #Shows the possible timezones for /etc/localtime
> timedatectl set-timezone *$timezonefile* #Sets /etc/localtime to the specified timezone.
> timedatectl set-ntp yes #Allows chronyc to sync the system time from ntp servers.
> date -d '+1 week' #Displays the current date + 1 week. Useful in command substitution.
> date -s '+1 week' #Sets the date and time ahead 1 week from current date and time.
> date -I #Output date in yyyy-mm-dd format. Can be used with -d option, like so: -I -d '+1 week'.
> chronyc #Enters the chronyc command prompt.
> chronyc allow *$host* #Allows specified host as an ntp client.
> chronyc sources -v #Shows time sources and what each field means.

LDAP Client

TL;DR: The RedHat exam only requires LDAP client configuration. In order to practice, however, it's a must to have an LDAP server. OpenLDAP (using **slapd** daemon), is what I used. In practice, however, Microsoft's Active Directory is by far the most commonly used LDAP service provider. The Linux LDAP client I used was **sssd**.

Notes:
> Helpful links and the sources for this material: Client configuration, Server configuration.
> The LDAP default port is 389. LDAPS (Secure LDAP using TLS) runs on port 636 by default.
> Linux LDAP client uses many complex configuration files. It isn't a good idea to manually edit these files. Authconfig, a command-line tool, authconfig-tui, a command-line textual user interface, and authconfig-gtk, a Gnome graphical utility, can all be used to edit these files.
> Install the sssd package (system security services daemon).
> If getent doesn't show a user, but the user shows up in ldapsearch, then restart **sssd**.
> To be able to log in, the user object needs to have a gidNumber and uidNumber attribute.

Commands:
> getent passswd *$ldapuser* #Checks to see if the specified user credentials is working on the client.
> ldapsearch -x #Checks the local LDAP cach/database.
> authconfig-gtk #Gnome graphical application for authconfig. May not be available during the test. Everything should be done via commandline only, but if the option is there, it might be ok.
> authconfig --help | grep ldap #Shows all authconfig options with ldap in the name. Super useful.
> authconfig --enableldap --enableldapauth --ldapserver="*$server.example.com*" --ldapbasedn="dc=example,dc=com" --update #The main command for setting up ldap client.
> authconfig --enableldaptls --update #After the command above, enables TLS encryption.

Files:
> /etc/openldap/cacerts/*.pem #Client side ldap cert. Public key.
> /etc/openldap/certs/*.pem #Server side ldap cert. Copy this key to the client location above otherwise TLS will fail to authenticate.

Sudo

TL;DR: Sudo stands for *super user do* or *super user do once*, depending on who you ask. It allows *specific users* to run *specific commands* as if they were a *specific user* all on the *specified hosts*. Although clearly there are a lot of options on how to use sudo, the most common situation is for a user to run a command as root.

Notes:
> Here is a great source on the fields of sudo.
> The syntax to put into /etc/sudoers is as follows: User Hosts=(Runas users) commands
or %Group Hosts=(Runas users) commands

> **Example:** In /etc/sudoers: kevin Srv1,Srv2(operator) ALL
#This allows the kevin user to run all commands as if he was the operator user only on hosts Srv1 and Srv2. %operators DB1=(root, dave) /bin/s*
#This allows all users that are a member of the operators group to run any executable that starts with s in the /bin directory on the DB1 server as the root, or dave user. The % before operators indicates that operators is a group, not a user.

Commands:
> visudo #Edits the /etc/sudoers file using the vi editor. Also is smart about the sudoers file syntax and will warn the user if the syntax is wrong. Very useful to prevent headaches.

Files:
> /etc/sudoers #File that the sudo command looks at to determine a user's sudo permissions.

Cron and At

TL;DR: Cron is the Linux job scheduler. Cron can be used to schedule jobs at specific intervals of time to run a task repeatedly (similar to Windows Task Scheduler). **Example:** Run this command once a week. The Linux At command is a scheduler that can be used to run a job once. **Example:** Run this command in 12 hours.

Notes:
> Each user has their own crontab (cron table). This means, unlike Windows, that each user can schedule their own jobs
> Cron has 6 fields; 5 timing fields and 1 field for an executable.
> The date command can be really useful to remember the fields in cron. The fields are the output of the date command but backwards.
> Cron and At both include the same HOME and USER variables as a normal shell but, Cron uses SHELL=/bin/sh, and At uses SHELL=/bin/bash
> Full paths make life easier in Cron.
> In cron, the command will only run when all 5 time fields are true. An asterix (*) will always be true.
> Cron fields: * * * * * *$command*

```
              | |  | | |
              | |  | | ----- Day of week (0 – 7) (Sunday=0 and 7)
              | |  | ------- Month (1 – 12)
              | |  --------- Day of month (1 – 31)
              | ----------- Hour (0 – 23)
              ------------- Minute (0 – 59)   #Ethically sourced from this nice website.
```

Example: 15 0 * * 6 ~/script.sh #Runs ~/script.sh every week on Saturday (6) at 15 minutes past midnight.
> Using multiple numbers (0,15,30,45) for one field is supported. Every other (*/2) or every 15th (*/15), etc. is also supported. Ranges, like 5-15 are supported as well.
> Instead of the 5 fields, "@reboot" can be used to run a job on startup.
> Unless redirected, all data from stdout and stderr streams will be sent to the owner's mail. This is for both At and Cron. Redirecting to a file is generally better practice because it's much easier than dealing with Linux mail.
> The cron service depends on **crond**, the cron daemon. If cron isn't behaving, a restart can sometimes fix it. The At service depends on **atd**, the At daemon.
> At timespecs are very flexible. Noon, now + 5 hours, 1230, 3 pm, tomorrow, 3pm tomorrow, and teatime are all accepted as timespecs.

Commands:
> crontab -l #Lists the current user's crontab.
> crontab -e #Enters a text editor to edit the current user's crontab.
> crontab -r #Removes all entries from the current user's crontab.
> at *$timespec* #Enters At prompt for the time specified.
> at> *$command* #Enters the command in to run when time time is reached.
> at> ^D #A Ctrl+D will break out of the at prompt and save the jobs entered.
> atq #Shows the future at queue.
> atrm *$jobnumber* #Removes from the at queue the job with the specified job number.

Files:
> /var/spool/cron/ #Location where every user's crontabs are stored.
> /var/spool/at/ #Location where every user's At jobs are stored.

Sysctl

TL;DR: Sysctl exists as a way to change kernel options/parameters. There are many, many options, and, like the Windows registry, they should only be changed with good reason. Best practice is to leave them alone unless something specifically needs a change. Backing up the parameters before a change never hurt either.

Example:
net/ipv4/ip_forward #Sysctl parameter that determines whether Linux functions as a router or not.

Commands:
> sysctl -a #Displays all parameters. There is a lot of output from this command, so using a pipe after is useful.

> sysctl -w *$parameter*="*$value*" #Changes the value of the specified parameter.
> sysctl -p *$file* #Restores parameter values from a file.

Nice and Renice

TL;DR: The nice and renice commands set and change the CPU priority given to a process. The "nicer" a process is, the lower CPU priority it has.

Notes:
> Niceness values span from -20 to 19. A process with -20 nice is mean, and will hog the CPU. A process with 19 is very nice (timid) and will not use the CPU unless it is the only process in the CPU run queue.
> The default niceness value for all processes is 0.
> The nice command is used to set the nice value for a new process, and the renice command is used to set the nice value for already running processes.

Commands:
> nice -n *$nicelevel* *$command* #Sets the nice level for the following command.
> renice -n *$nicelevel* *$pid* #Changes the process nice level that belongs to the specified pid.
> ps -eo pid,ni,comm #Will show the pid, nice level, and command for all processes on the system.

Iptables/Ip6tables

TL;DR: Iptables is one of many rule based firewall softwares to allow and block specific network traffic.

Notes:
> The Redhat objectives specify that either firewall-cmd (**firewalld**), or **iptables** can be used for firewall configuration. Since I know iptables already, I have decided to use them. Firewall-cmd is not in these notes, but it wouldn't hurt to study it.
> This is a great resource for iptables beginners.
> Iptables filter table uses 3 different chains by default. INPUT, OUTPUT, and FORWARD. Each chain has its own rules.
> You can add more chains for things, but it gets clusterfucked easily. Stick to the 3 defaults if you can.
> There are 3 options on what to do with data coming in: ACCEPT, REJECT, or DROP. Accepting is what it sounds like, Rejecting will not allow traffic in, but will send a notification message back to the source, while dropping is the stealthiest, and will silently drop packets with no warning back to the source.
> Iptables processes a packet like this: First, a packet is sent to the chain it belongs to. For example, if a packet is incoming, it is sent to the INPUT chain. If it's going out, then it belongs to the OUTPUT chain. Once in a chain, the packet goes down the list of rules checking if any rule matches the packet. Once a matching rule is encountered, the action with the rule is taken on the packet. If a packet does not match any of the rules in the chain, the chain default behavior is used. This is shown in the chain policy, and is set to ALLOW by default.

> Remember ip6tables exists too. It's too easy to forget this, and everything done to iptables must be done to ip6tables as well for IPv6 filtering. This includes dealing with the **ip6tables** service, and of course, adding rules.

Commands:
> yum install iptables-services #Installs the systemctl service controller for iptables.
> systemctl start **iptables** #Start iptables for the current boot.
> systemctl enable **iptables** #Enables iptables persistently. It will be started whenever Linux boots.
> man iptables-extensions #Shows some information on more advanced iptables options. Stuff like connection tracking and advanced packet matching. Useful for beyond basic packet filtering.

Iptables Flags:
> -t *$table* #Uses the specified table to do operations on. Tables include filter, nat, mangle, raw, and security. If no table is specified, uses the filter table.
> -n #Numeric output. Shows port numbers and IP's instead of services and hostnames.
> -F #Flushes (deletes) all current iptables rules.
> -L #Lists all current rules.
> -A *$chain* #Append a rule to the specified chain. These are INPUT, FORWARD, and OUPUT normally, but can be more with user created chains or chains from other tables.
> -p [tcp|udp|icmp|all] #Matches the specified protocol.
> --dport [*$portnumber*|*$protocolname*] #Specifies the destination port number or service the rule applies to. This option requires a protocol to be specified using the -p option.
> -j [DROP|REJECT|ACCEPT] #Decide what to do with a matching packet.
> -D #Deletes the matching rule. Does the opposite of -A. Like -I, -D can also use a chain and rule number, but to delete a rule instead of insert.
> -I *$chain* *$rulenumber* #Insert. Use instead of -A with a number after to specify where to insert the rule.
> -v #Verbose listing. Shows packets and byte amounts for chains and rules.
> --policy *$chain* [DROP|REJECT|ACCEPT] #Sets the chain's behavior when a packet gets to the bottom of the chain without matching a rule.

Examples:
iptables -nL #Lists iptables. Most common iptables command
iptables -F #Removes all rule entries from the iptables
iptables -A INPUT -p tcp --dport 22 -j ACCEPT #Appends a rule to the INPUT chain accepting tcp destination port 22.
iptables -D INPUT -p tcp --dport 22 -j ACCEPT #Removes the rule above.
iptables -I OUTPUT 3 --dport http -j DROP #Adds a rule between the current 2nd and 3rd rule in the OUTPUT chain that drops destination port 80.
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT #Stateful firewall rule. Allows all incoming traffic ONLY if it was initiated by the local host.

End

Version history:

> 1.0.0 November 4, 2017 #Initial release.
> 1.0.1 December 19, 2017 #Stateful iptables firewall rule added. Preface edited.
> 1.0.2 February 20, 2018 #Added Cron variables, ip6tables, more iptables options, misc spelling and formatting.
> 1.0.3 April 6, 2018 #Removed alternate breakin method, edited yum repo file, added rpm commands.
> 1.1.0 June 12, 2018 #Added KVM and Virsh section, added more to yum section, edited root breakin, xfs noshrink in lvm, various typos.
> 1.1.1 June 29, 2018 #Passed the RHCSA. Editing some typos and formatting errors.