



Red Hat System Administration I

Dr. Eng. Gamal Selim

Lecture 08

Agenda

- **Chapter 13: Access Linux File Systems**

ACCESSING LINUX FILE SYSTEMS

GOAL

To access and inspect existing file systems on a Red Hat Enterprise Linux system.

OBJECTIVES

- Identify the file system hierarchy.
- Access the contents of file systems.
- Use hard links and symlinks to make multiple names.
- Search for files on mounted file systems.

SECTIONS

- Identifying File Systems and Devices (and Practice)
- Mounting and Unmounting File Systems (and Practice)
- Making Links Between Files (and Practice)
- Locating Files on the System (and Practice)

IDENTIFYING FILE SYSTEMS AND DEVICES

OBJECTIVES

After completing this section, students should be able to identify a directory in the file system hierarchy and what storage device it is stored on.

STORAGE MANAGEMENT CONCEPTS

A file system is an organized structure of data-holding files and directories residing on a storage device, such as a physical disk or partition. The file system hierarchy discussed earlier assembles all the file systems into one tree of directories with a single root, the `/` directory. The advantage here is that the existing hierarchy can be extended at any time by adding a new disk or partition containing a supported file system to add disk space anywhere in the file system tree. The process of adding a new file system to the existing directory tree is called *mounting*. The directory where the new file system is mounted is referred to as a *mount point*. This is a fundamentally different concept than that used on a Microsoft Windows system, where a new file system is represented by a separate drive letter.

Hard disks and storage devices are normally divided up into smaller chunks called *partitions*. A partition is a way to compartmentalize a disk. Different parts of it can be formatted with different file systems or used for different purposes. For example, one partition could contain user home directories while another could contain system data and logs. If a user fills up the home directory partition with data, the system partition may still have space available. Placing data in two separate file systems on two separate partitions helps in planning data storage.

Storage devices are represented by a special file type called *block device*. The block device is stored in the **/dev** directory. In Red Hat Enterprise Linux, the first SCSI, PATA/SATA, or USB hard drive detected is **/dev/sda**, the second is **/dev/sdb**, and so on. This name represents the whole drive. The first primary partition on **/dev/sda** is **/dev/sda1**, the second partition is **/dev/sda2**, and so on.

A long listing of the **/dev/vda** device file on serverX reveals its special file type as **b**, which stands for block device:

```
[student@serverX ~]$ ls -l /dev/vda  
brw-rw----. 1 root disk 253, 0 Mar 13 08:00 /  
dev/vda
```

EXAMINING FILE SYSTEMS

To get an overview about the file system mount points and the amount of free space available, run the **df** command. When the **df** command is run without arguments, it will report total disk space, used disk space, and free disk space on all mounted regular file systems. It will report on both local and remote systems and the percentage of the total disk space that is being used.

Display the file systems and mount points on the serverX machine.

```
[student@serverX ~]$ df
Filesystem      1K-blocks    Used Available Use% Mounted on
/dev/vda1        6240256 4003760   2236496   65% /
devtmpfs         950536      0    950536    0% /dev
tmpfs            959268     80    959188    1% /dev/shm
tmpfs            959268    2156    957112    1% /run
tmpfs            959268      0    959268    0% /sys/fs/cgroup
```

Show a report on the file systems on the serverX machine with all units converted to human-readable format:

```
[student@serverX ~]$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        6.0G  3.9G  2.2G  65% /
devtmpfs         929M   0    929M   0% /dev
tmpfs            937M   80K   937M   1% /dev/shm
tmpfs            937M  2.2M   935M   1% /run
tmpfs            937M   0    937M   0% /sys/fs/cgroup
```

Show a disk usage report for the **/root** directory on serverX:

```
[root@serverX ~]# du /root
4 /root/.ssh
4 /root/.cache/dconf
4 /root/.cache
4 /root/.dbus/session-bus
4 /root/.dbus
```

Show a disk usage report in human-readable format for the **/var/log** directory on serverX:

```
[root@serverX ~]# du -h /var/log
...
4.9M /var/log/sa
68K /var/log/prelink
0 /var/log/qemu-ga
14M /var/log
```

MOUNTING AND UNMOUNTING FILE SYSTEMS

OBJECTIVES

After completing this section, students should be able to access the contents of file systems by adding and removing file systems from the file system hierarchy.

MOUNTING FILE SYSTEMS MANUALLY

A file system residing on a SATA/PATA or SCSI device needs to be mounted manually to access it. The **mount** command allows the root user to manually mount a file system. The first argument of the **mount** command specifies the file system to mount. The second argument specifies the target directory where the file system is made available after mounting it. The target directory is referred to as a mount point.

The **mount** command expects the file system argument in one of two different ways:

- The device file of the partition holding the file system, residing in **/dev**.
- The *UUID*, a universal unique identifier of the file system.

The **blkid** command gives an overview of existing partitions with a file system on them and the UUID of the file system, as well as the file system used to format the partition.

```
[root@serverX ~]# blkid  
/dev/vda1: UUID="46f543fd-78c9-4526-a857-244811be2d88" TYPE="xfs"
```

Mount by device file of the partition that holds the file system.

```
[root@serverX ~]#  
mount /dev/vdb1 /mnt/mydata
```

Mount the file system by universal unique id, or the UUID, of the file system.

```
[root@serverX ~]#  
mount UUID="46f543fd-78c9-4526-a857-244811be2d88" /mnt/mydata
```

UNMOUNTING FILE SYSTEMS

To unmount a file system, the **umount** command expects the mount point as an argument.

Change to the **/mnt/mydata** directory. Try to unmount the device mounted on the **/mnt/mydata** mount point. It will fail.

```
[root@serverX ~]# cd /mnt/mydata[root@serverX mydata]# umount /mnt/mydata
umount: /mnt/mydata: target is busy.
```

(In some cases useful info about processes that use
the device is found by `lsof(8)` or `fuser(1)`)

Unmounting is not possible if the mount point is accessed by a process. For **umount** to be successful, the process needs to stop accessing the mount point.

The **lsof** command lists all open files and the process accessing them in the provided directory. It is useful to identify which processes currently prevent the file system from successful unmounting.

```
[root@serverX mydata]# lsof /mnt/mydata
```

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE/OFF	NODE	NAME
bash	1593	root	cwd	DIR	253,2	6	128	/mnt/mydata
lsof	2532	root	cwd	DIR	253,2	19	128	/mnt/mydata
lsof	2533	root	cwd	DIR	253,2	19	128	/mnt/mydata

MAKING LINKS BETWEEN FILES

OBJECTIVES

After completing this section, students should be able to use hard links and soft links to make multiple names point to the same file.

MANAGING LINKS BETWEEN FILES

Creating hard links

A hard link is a new directory entry with a reference to an existing file on the file system. Every file in a file system has one hard link by default. To save space, instead of copying, a new hard link can be created to reference the same file. A new hard link either needs to have a different file name, if it is created in the same directory as the existing hard link, or it needs to reside in a different directory. All hard links pointing to the same file have the same permissions, link count, user/group ownerships, time stamps, and file content. Hard links pointing to the same file content need to be on the same file system.

The **ls -l** shows the hard link count after the permissions and before the owner of a file.

```
[root@serverX ~]# echo "Hello World" > newfile.txt
[root@serverX ~]# ls -l
newfile.txt
-rw-r--r--. 1 root root 0 Mar 11 19:19 newfile.txt
```

The command **ln** creates new hard links to existing files. The command expects an existing file as the first argument, followed by one or more additional hard links. The hard links can reside anywhere as long as they are on the same file system as the existing file. After a new hard link is created, there is no way to tell which of the existing hard links is the original one.

Create a hard link **newfile-link2.txt** for the existing file **newfile.txt** in the **/tmp** directory.

```
[root@serverX ~]# ln newfile.txt /tmp/newfile-hlink2.txt [root@serverX ~]# ls -l  
newfile.txt /tmp/newfile-hlink2.txt  
-rw-rw-r--. 2 root root 12 Mar 11 19:19 newfile.txt  
-rw-rw-r--. 2 root root 12 Mar 11 19:19 newfile-hlink2.txt
```

Even if the original file gets deleted, the content of the file is still available as long as at least one hard link exists.

```
[root@serverX ~]# rm -f newfile.txt
[root@serverX ~]# ls -l /tmp/newfile-link2.txt
-rw-rw-r--. 1 root root 12 Mar 11 19:19 /tmp/newfile-link2.txt
[root@serverX ~]# cat /tmp/newfile-link2.txt
Hello World
```

Creating soft links

The **ln -s** command creates a soft link, which is also called a "symbolic link". A soft link is not a regular file, but a special type of file that points to an existing file or directory. Unlike hard links, soft links can point to a directory, and the target to which a soft link points can be on a different file system.

```
[root@serverX ~]# ln -s /root/newfile-link2.txt /tmp/newfile-  
symlink.txt [root@serverX ~]# ls -l newfile-link2.txt /tmp/newfile-  
symlink.txt lrwxrwxrwx. 1 root root 11 Mar 11 20:59 /tmp/newfile-symlink.txt -> /  
root/newfile-link2.txt  
-rw-rw-r--. 1 root root 12 Mar 11 19:19 newfile-link2.txt
```

When the original file gets deleted, the soft link is still pointing to the file but the target is gone. A soft link pointing to a missing file is called a "dangling soft link."

```
[root@serverX ~]# rm -f newfile-link2.txt [root@serverX ~]# ls -l /tmp/newfile-  
symlink.txt lrwxrwxrwx. 1 root root 11 Mar 11 20:59 /tmp/newfile-symlink.txt ->  
newfile-link2.txt  
[root@serverX ~]# cat /tmp/newfile-symlink.txt  
cat: /tmp/newfile-symlink.txt: No such file or directory
```

A soft link can point to a directory. The soft link then acts like a directory. Changing to the soft link directory with **cd** works as expected.

Create a soft link **/root/configfiles** pointing to the **/etc** directory.

```
[root@serverX ~]# ln -s /etc /root/configfiles [root@serverX ~]# cd /root/  
configfiles [root@serverX configfiles]# pwd  
/root/configfiles
```

LOCATING FILES ON THE SYSTEM

OBJECTIVES

After completing this section, students should be able to search for files on mounted file systems using **find** and **locate**.

TOOLS FOR FINDING FILES

A system administrator needs tools for searching files matching certain criteria on the file system. This section discusses two commands that can search files in the file system. The **locate** command searches a pregenerated database for file names or file paths and returns the results instantly. The **find** command searches the file system in real time by crawling through the file system.

LOCATING FILES BY NAME WITH **locate**

The **locate** command returns search results based on file name or path from the **locate** database. The database stores file name and path information.

When searching entries as a regular user, results are returned only for where the user invoking the **locate** search has read permissions on the directory tree that contains the matching element.

Search for files with "passwd" in the name or path in directory trees readable by user student on serverX.

```
[student@serverX ~]$ locate passwd
/etc/passwd
/etc/passwd-
/etc/pam.d/passwd
/etc/security/opasswd
/usr/bin/gpasswd
/usr/bin/grub2-mkpasswd-pbkdf2
/usr/bin/lpasswd
/usr/bin/passwd
/usr/bin/userpasswd
/usr/bin/vino-passwd
/usr/bin/vncpasswd
```

Results are returned even when the file name or path is only a partial match to the search query.

```
[root@serverX ~]# locate image  
/home/myuser/boot.image  
/home/someuser/my_family_image.png  
/home/student/myimages-vacation/picture.png
```

The **-i** option performs a case-insensitive search. With this option, all possible combinations of upper- and lowercase letters match the search.

```
[student@serverX ~]$ locate -i messages  
...
```

The **-n** option limits the number of returned search results by **locate**. The following example limits the search results returned by **locate** to the first five matches.

```
[student@serverX ~]$ locate -n 5 snow.png
/usr/share/icons/HighContrast/16x16/status/weather-snow.png
/usr/share/icons/HighContrast/22x22/status/weather-snow.png
/usr/share/icons/HighContrast/24x24/status/weather-snow.png
/usr/share/icons/HighContrast/256x256/status/weather-snow.png
/usr/share/icons/HighContrast/32x32/status/weather-snow.png
```

SEARCHING FOR FILES WITH FIND

The **find** command performs a real-time search in the local file systems to find files that match the criteria of the command-line arguments. The **find** command is looking at files in the file system as your user account. The user invoking the **find** command must have read and execute permission on a directory to examine its contents.

The first argument to the **find** command is the directory to search. If the directory argument is omitted, **find** will start the search in the current directory and look for matches in any of the subdirectories.

To search the home directory of user student, give **find** a starting directory of **/home/student**. To search the entire system, provide a starting directory of **/**.

The **-name** option followed by a file name looks up files matching the given file name and returns all exact matches. To search for files named **sshd_config** in the **/** directory and all subdirectories on serverX, run:

```
[root@serverX ~]# find / -name sshd_config
```

The following example searches for files in the `/` directory on serverX that end in `.txt`:

```
[root@serverX ~]# find / -name '*.txt'
/etc/pki/nssdb/pkcs11.txt
/etc/brltty/brl-lt-all.txt
/etc/brltty/brl-mb-all.txt
/etc/brltty/brl-md-all.txt
/etc/brltty/brl-mn-all.txt
...
```

To search for files in **/etc/** that contain **pass** anywhere in their names on serverX, run:

```
[root@serverX ~]# find /etc -name '*pass*'
/etc/passwd
/etc/passwd-
/etc/fonts/conf.d/60-overpass.conf
/etc/selinux/targeted/modules/active/modules/passenger.pp
/etc/security/opasswd
/etc/pam.d/passwd
/etc/pam.d/password-auth-ac
/etc/pam.d/password-auth
/etc/pam.d/gdm-password
```

To perform a case-insensitive search for a given file name, use the **-iname** option, followed by the file name to search. To search case-insensitively for files that have **messages** in their names in the **/** directory on serverX, run:

```
[root@serverX ~]# find / -iname '*messages*'
/var/log/messages
/usr/lib64/python2.7/site-packages/orca/notification_messages.py
/usr/lib64/python2.7/site-packages/orca/notification_messages.pyc
/usr/lib64/python2.7/site-packages/orca/notification_messages.pyo
/usr/share/locale/aa/LC_MESSAGES
/usr/share/locale/ab/LC_MESSAGES
/usr/share/locale/ace/LC_MESSAGES
```

find can search for files based on their ownership or permissions. Useful options when searching by owner are **-user** and **-group**, which search by name, and **-uid** and **-gid**, which search by ID.

Search for files owned by the user *student* in the **/home/student** directory on serverX.

```
[student@serverX ~]$ find -user student
.
./ .bash_logout
./ .bash_profile
./ .bashrc
./ .ssh
...
```

Search for files owned by the group *student* in the **/home/student** directory on serverX.

```
[student@serverX ~]$ find -group student
.
./bash_logout
./bash_profile
./bashrc
./ssh
...
```

Search for files owned by user ID *1000* in the **/home/student** directory on serverX.

```
[student@serverX ~]$ find -uid 1000
.
./bash_logout
./bash_profile
./bashrc
./ssh
...
```

Search for files owned by group ID *1000* in the **/home/student** directory on serverX.

```
[student@serverX ~]$ find -gid 1000
```

```
.  
./bash_logout  
./bash_profile  
./bashrc  
./ssh  
...
```

Search for files owned by user *root* and group *mail* on the serverX machine.

```
[root@serverX ~]# find / -user root -group mail  
/var/spool/mail  
/var/spool/mail/root
```

The **-perm** option is used to look for files with a particular set of permissions. Permissions can be described as octal values, with some combination of 4, 2, and 1 for read, write, and execute. Permissions can be preceded by a **/** or **-** sign.

A numeric permission preceded by **/** will match files that have at least one bit of user, group, or other for that permission set. A file with permissions **r--r--r--** does not match **/222**, but one with **rw-r--r--** does. A **-** sign before a permission means that all three instances of that bit must be on, so neither of the previous examples would match, but something like **rw-rw-rw-** would.

To use a more complex example, the following command would match any file for which the user has read, write, and execute permissions, members of the group have read and write permissions, and others have read-only access:

```
[root@serverX ~]#  
find /home -perm 764
```

The **find** command can look up files that match a size specified with the **-size** option, followed by a numerical value and the unit.

Units to be used with the **-size** option are:

- k, for kilobyte
- M, for megabyte
- G, for gigabyte

Search for files with a size of *exactly* 10 megabytes.

```
[student@serverX ~]$  
find -size 10M
```

```
[student@serverX ~]$  
find -size +10G
```

List all files with a size *less than* 10 kilobytes.

```
[student@serverX ~]$  
find -size -10k
```

The **-mmin** option, followed by the time in minutes, searches for all files that had their content changed at exactly the given time in the past.

To find all files that had their file content changed exactly 120 minutes ago on serverX, run:

```
[root@serverX ~]#  
find / -mmin 120
```

The **+** modifier in front of the amount of minutes looks for all files in the **/** that have been modified more than 200 minutes ago.

```
[root@serverX ~]#  
find / -mmin +200
```

The **-type** option limits the search scope to a given file type, such as:

- **f**, for regular file
- **d**, for directory
- **l**, for soft link
- **b**, for block device

Search for all directories in the **/etc** folder on serverX.

```
[root@serverX ~]# find /etc -type d
/etc
/etc/tmpfiles.d
/etc/systemd
/etc/systemd/system
/etc/systemd/system/getty.target.wants
...
```

Search for all soft links on the serverX system.

```
[root@serverX ~]#
find / -type l
```

Generate a list of all block devices in the **/dev** directory on serverX:

```
[root@serverX ~]# find /dev -type b
/dev/vda1
/dev/vda
```

End of Chapter 13

*Thank
you*

