ACCESSING LINUX FILE SYSTEMS

Objectives

- Explain what a block device is, interpret the file names of storage devices, and identify the storage device used by the file system for a particular directory or file.
- Access file systems by attaching them to a directory in the file system hierarchy.
- Identify file systems by attaching them to a directory in a file system hierarchy.
- Search for files on mounted file systems using the **find** and **locate** commands.

IDENTIFYING FILE SYSTEMS AND DEVICES

1. STORAGE MANAGEMENT CONCEPTS

Files on a Linux server are accessed through the file-system hierarchy, a single inverted tree of directories. This file system hierarchy is assembled from file systems provided by the storage devices available to your system. Each file system is a storage device that has been formatted to store files.

In a sense, the Linux file-system hierarchy presents a collection of file systems on separate storage devices as if it were one set of files on one giant storage device that you can navigate. Much of the time, you do not need to know which storage device a particular file is on, you just need to know the directory that file is in.

Sometimes, however, it can be important. You might need to determine how full a storage device is and what directories in the file-system hierarchy are affected. There might be errors in the logs from a storage device, and you need to know what file systems are at risk. You could just want to create a hard link between two files, and you need to know if they are on the same file system to determine if it is possible.

File Systems and Mount Points

To make the contents of a file system available in the file-system hierarchy, it must be mounted on an empty directory. This directory is called a mount point. Once mounted, if you use ls to list that directory, you will see the contents of the mounted file system, and you can access and use those files normally. Many file systems are automatically mounted as part of the boot process.

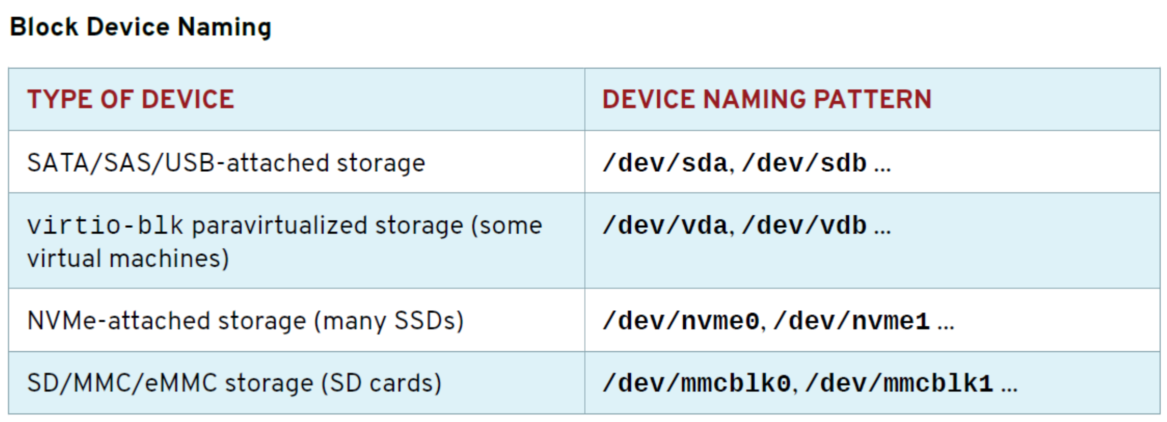
If you have only worked with Microsoft Windows drive letters, this is a fundamentally different concept. It is somewhat similar to the NTFS mounted folders feature.

File Systems, Storage, and Blocked Devices

To Low-level access to storage devices in Linux is provided by a special type of file called a block device. These block devices must be formatted with a file system before they can be mounted.

Block device files are stored in the **/dev** directory, along with other device files. Device files are created automatically by the operating system. In Red Hat Enterprise Linux, the first SATA/PATA, SAS, SCSI, or USB hard drive detected is called **/dev/sda**, the second is **/dev/sdb,** and so on. These names represent the entire hard drive.

Other types of storage will have other forms of naming.



Block Device Naming

TYPE OF DEVICE	DEVICE NAMING PATTERN
SATA/SAS/USB-attached storage	`/dev/sda`, `/dev/sdb` ...
`virtio-blk` paravirtualized storage (some virtual machines)	`/dev/vda`, `/dev/vdb` ...
NVMe-attached storage (many SSDs)	`/dev/nvme0`, `/dev/nvme1` ...
SD/MMC/eMMC storage (SD cards)	`/dev/mmcblk0`, `/dev/mmcblk1` ...

Disk Partitions

Normally, you do not make the entire storage device into one file system. Storage devices are typically divided up into smaller chunks called *partitions.*

Partitions allow you to compartmentalize a disk: the various partitions can be formatted with different file systems or used for different purposes. For example, one partition can contain user home directories while another can contain system data and logs. If a user fills up the home directory partition with data, the system partition may still have space available.

Partitions are block devices in their own right. On SATA-attached storage, the first partition on the first disk is **/dev/sda1**. The third partition on the second disk is **/dev/sdb3,** and so on.

Paravirtualized storage devices have a similar naming system.

An NVMe-attached SSD device names its partitions differently. In that case, the first partition on the first disk is **/dev/nvme0p1**. The third partition on the second disk is **/dev/nvme1p3**, and so on. SD or MMC cards have a similar naming system.

A long listing of the **/dev/sda1** device file on host reveals its special file type as b, which stands for block device:

```
[student@localhost ~] ls -l /dev/sda1
brw-rw----. 1 root disk 8, 1 Dec 26 10:37 /dev/sda1
```

Logical Volumes

Another way of organizing disks and partitions is with logical volume management (LVM). With LVM, one or more block devices can be aggregated into a storage pool called a volume group. Disk space in the volume group is then parceled out to one or more logical volumes, which are the functional equivalent of a partition residing on a physical disk.

The LVM system assigns names to volume groups and logical volumes upon creation. LVM creates a directory in /dev that matches the group name and then creates a symbolic link within that new directory with the same name as the logical volume. That logical volume file is then available to be mounted. For example, if a volume group is called myvg and the logical volume within it is called **mylv**, then the full path name to the logical volume device file is **/dev/myvg/mylv**.

2. EXAMINING FILE SYSTEMS

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

The following example displays the file systems and mount points on host.

```
[user@host ~]$ df

Filesystem   1K-blocks   Used   Available   Use%   Mounted on
devtmpfs     912584      0      912584      0%     /dev
tmpfs        936516      0      936516      0%     /dev/shm
tmpfs           936516          16812  919704           2%       /run
tmpfs           936516          0          936516           0%       /sys/fs/cgroup
/dev/vda3       8377344         1411332 6966012        17%      /
/dev/vda1       1038336         169896   868440         17%      /boot
tmpfs           187300          0          187300            0%       /run/user/1000
```

The partitioning on the host system shows two physical file systems, which are mounted on **/** and **/boot**. This is common for virtual machines. The tmpfs and devtmpfs devices are file systems in system memory. All files written into tmpfs or devtmpfs disappear after system reboot.

To improve readability of the output sizes, there are two different human-readable options: **-h** or **-H**. The difference between these two options is that -h reports in KiB (210), MiB (220), or GiB (230), while the -H option reports in SI units: KB (103), MB (106), or GB (109). Hard drive manufacturers usually use SI units when advertising their products.

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

[user@host ~]$ df -h

```
Filesystem      Size            Used    Avail   Use%    Mounted on
devtmpfs        892M            0       892M    0%      /dev
tmpfs           915M            0       915M    0%      /dev/shm
tmpfs           915M            17M     899M     2%     /run
tmpfs           915M            0       915M    0%      /sys/fs/cgroup
/dev/vda        3 8.0G          1.4G    6.7G    17%     /
/dev/vda1       1014M           166M    849M    17%     /boot
tmpfs           183M            0       183M    0%      /run/user/1000
```

For more detailed information about space used by a certain directory tree, use the du command. The **du** command has **-h** and **-H** options to convert the output to human-readable format. The **du** command shows the size of all files in the current directory tree recursively.

Show a disk usage report for the /usr/share directory on host:

```
[root@host ~]# du /usr/share
...output omitted...
176     /usr/share/smartmontools
184     /usr/share/nano
8       /usr/share/cmake/bash-completion
8       /usr/share/cmake
356676 /usr/share
```

Show a disk usage report in human-readable format for the /usr/share directory on host:

```
[root@host ~]# du -h /var/log
...output omitted...
176K    /usr/share/smartmontools
184K    /usr/share/nano
8.0K    /usr/share/cmake/bash-completion
8.0K    /usr/share/cmake
369M    /usr/share
```

MOUNTING AND UNMOUNTING FILE SYSTEMS

1. MOUNTING FILE SYSTEMS MANUALLY

A file system residing on a removable storage device needs to be mounted in order 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 directory to use as the mount point in the file-system hierarchy.

There are two common ways to specify the file system on a disk partition to the mount command:

- With the name of the device file in /dev containing the file system.
- With the UUID written to the file system, a universally-unique identifier.

Mounting a device is relatively simple. You need to identify the device you want to mount, make sure the mount point exists, and mount the device on the mount point.

Identifying the Block Device

A hot-pluggable storage device, whether a hard disk drive (HDD) or solid-state device (SSD) in a server caddy, or a USB storage device, might be plugged into a different port each time they are attached to a system.

Use the **lsblk** command to list the details of a specified block device or all the available devices.

```
[root@host ~]# lsblk

NAME        MAJ:MIN   RM   SIZE   RO   TYPE   MOUNTPOINT
vda         253:0     0    12G    0    disk
├─vda1      253:1     0     1G    0    part   /boot
├─vda2      253:2     0     1G    0    part   [SWAP]
└─vda3      253:3     0    11G    0    part   /
vdb         253:16    0    64G    0    disk
└─vdb1      253:17    0    64G    0    part
```

If you know that you just added a 64 GB storage device with one partition, then you can guess from the preceding output that **/dev/vdb1** is the partition that you want to mount.

Mounting by Block Device Name

The following example mounts the file system in the **/dev/vdb1** partition on the directory **/mnt/data.**

```
[root@host ~]# mount /dev/vdb1 /mnt/data
```

To mount a file system, the destination directory must already exist. The **/mnt** directory exists by default and is intended for use as a temporary mount point.

You can use **/mnt** directory, or better yet create a subdirectory of **/mnt** to use as a temporary mount point, unless you have a good reason to mount it in a specific location in the file-system hierarchy.

This approach works fine in the short run. However, the order in which the operating system detects disks can change if devices are added to or removed from the system. This will change the device name associated with that storage device. A better approach would be to mount by some characteristic built into the file system.

Mounting by File-system UUID

One stable identifier that is associated with a file system is its UUID, a very long hexadecimal number that acts as a universally-unique identifier. This UUID is part of the file system and remains the same as long as the file system is not recreated.

The **lsblk -fp** command lists the full path of the device, along with the UUIDs and mount points, as well as the type of file system in the partition. If the file system is not mounted, the mount point will be blank.

```
[root@host ~]# lsblk –fp
```

```
NAME        FSTYPE LABEL    UUID                                    MOUNTPOINT
/dev/vda
├─/dev/vda1   xfs           23ea8803-a396-494a-8e95-1538a53b821c    /boot
├─/dev/vda2   swap          cdf61ded-534c-4bd6-b458-cab18b1a72ea    [SWAP]
└─/dev/vda3   xfs           44330f15-2f9d-4745-ae2e-20844f22762d    /
/dev/vdb
└─/dev/vdb1   xfs           46f543fd-78c9-4526-a857-244811be2d88
```

Mount the file system by the UUID of the file system.

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

2. AUTOMATIC MOUNTING OF REMOVABLE STORAGE

If you are logged in and using the graphical desktop environment, it will automatically mount any removable storage media when it is inserted.

The removable storage device is mounted at **/run/media/USERNAME/LABEL** where *USERNAME* is the name of the user logged into the graphical environment and *LABEL* is an identifier, often the name given to the file system when it was created if one is available.

Before removing the device, you should unmount it manually.

3. UNMOUNTING FILE SYSTEMS

The shutdown and reboot procedures unmount all file systems automatically. As part of this process, any file system data cached in memory is flushed to the storage device thus ensuring that the file system suffers no data corruption.

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

```
[root@host ~]# umount /mnt/data
```

Unmounting is not possible if the mounted file system is in use. For the **umount** command to succeed, all processes needs to stop accessing data under the mount point.

In the example below, the **umount** fails because the file system is in use (the shell is using **/mnt/data** as its current working directory), generating an error message.

```
[root@host ~]# cd /mnt/data
```

```
[root@host data]# umount /mnt/data
umount: /mnt/data: target is busy.
```

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@host data]# lsof /mnt/data
COMMAND PID     USER FD    TYPE DEVICE  SIZE/OFF NODE NAME
bash        1593  root  cwd  DIR   253,17  6          128    /mnt/data
lsof        2532  root  cwd  DIR   253,17  19         128    /mnt/data
lsof        2533  root  cwd  DIR   253,17  19         128    /mnt/data
```

Once the processes are identified, an action can be taken, such as waiting for the process to complete or sending a SIGTERM or SIGKILL signal to the process. In this case, it is sufficient to change the current working directory to a directory outside the mount point.

```
[root@host data]# cd
[root@host ~]# umount /mnt/data
```

Tasks: Mounting and Unmounting File Systems

1. Use the ssh command to log in to localhost as the student user.

```
[student@localhost ~]$ ssh student@servera
[student@servera ~]$
```

2. Mount the **/dev/sda2** partition by UUID at the newly created mount point **/mnt/newspace.**

a. Use the su - command to switch to root, as the root user can only manually mount a device.

```
[student@localhost ~]$ su -
 [root@servera ~]#
```

b. Create the /mnt/newspace directory.

```
[root@servera ~]# mkdir /mnt/newspace
```

c. Use the **lsblk** command with the **-fp** option to discover the UUID of the device, **/dev/sda2**.

```
[root@localhost ~]# lsblk -fp /dev/sda2
NAME  FSTYPE        LABEL  UUID          MOUNTPOINT
/dev/sda2
└─/dev/sda2              xfs     a04c511a-b805-4ec2-981f-42d190fc9a65
```

d. Mount the file system by using UUID on the **/mnt/newspace** directory. Replace the UUID with that of the **/dev/sda2** disk from the previous command output.

```
[root@localhost ~]# mount UUID="a04c511a-b805-4ec2-981f-42d190fc9a65"
/mnt/newspace
```

e. Verify that the /dev/sda2 device is mounted on the /mnt/newspace directory.

```
[root@servera ~]# lsblk -fp /dev/sda2
NAME FSTYPE LABEL     UUID                                        MOUNTPOINT
/dev/sda2
└─/dev/sda2     xfs       a04c511a-b805-4ec2-981f-42d190fc9a65   /mnt/newspace
```

3. Change to the **/mnt/newspace** directory and create a new directory, /mnt/newspace/newdir, with an empty file, /mnt/newspace/newdir/newfile.

a. Change to the /mnt/newspace directory.

```
[root@localhost ~]# cd /mnt/newspace
```

b. Create a new directory, /mnt/newspace/newdir.

```
[root@localhost newspace]# mkdir newdir
```

c. Create a new empty file, /mnt/newspace/newdir/newfile.

```
[root@localhost newspace]# touch newdir/newfile
```

4. Unmount the file system mounted on the /mnt/newspace directory.

a. Use the umount command to unmount /mnt/newspace while the current directory on the shell is still /mnt/newspace. The umount command fails to unmount the device.

```
[root@localhost newspace]# umount /mnt/newspace
umount: /mnt/newspace: target is busy.
```

b. Change the current directory on the shell to /root.

```
[root@servera newspace]# cd
[root@servera ~]#
```

c. Now, successfully unmount /mnt/newspace.

5.
```
[root@localhost ~]# umount /mnt/newspace
```

Exit from a.

```
[root@ localhost ~]# exit
logout
[student@ localhost ~]$ exit
logout
Connection to localhost closed.
```

LOCATING FILES ON THE SYSTEM

1. SEARCHING FOR FILES

A system administrator needs tools to search for files matching certain criteria on the file system.

- The **locate** command searches a pregenerated index for file names or file paths and returns the results instantly.
- The **find** command searches for files in real time by crawling through the file-system hierarchy.

2. LOCATING FILES BY NAME

The **locate** command finds files based on the name or path to the file. It is fast because it looks up this information from the mlocate database. However, this database is not updated in real time, and it must be frequently updated for results to be accurate. This also means that locate will not find files that have been created since the last update of the database.

The **locate** database is automatically updated every day. However, at any time the root user can issue the **updatedb** command to force an immediate update.

```
[root@host ~]# updated
```

The locate command restricts results for unprivileged users. In order to see the resulting file name, the user must have search permission on the directory in which the file resides.

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

```
[user@host ~]$ locate passwd
/etc/passwd
/etc/passwd-
/etc/pam.d/passwd
```

```
/etc/security/opasswd
/usr/bin/gpasswd
/usr/bin/grub2-mkpasswd-pbkdf2
/usr/bin/lppasswd
/usr/bin/passwd
...output omitted...
```

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

```
[root@host ~]# locate image
/etc/selinux/targeted/contexts/virtual_image_context
/usr/bin/grub2-mkimage
/usr/lib/sysimage
/usr/lib/dracut/dracut.conf.d/02-generic-image.conf
/usr/lib/firewalld/services/ovirt-imageio.xml
/usr/lib/grub/i386-pc/lnxboot.image
...output omitted...
```

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

```
[user@host ~]$ locate -i messages
...output omitted...
/usr/share/vim/vim80/lang/zh_TW/LC_MESSAGES
/usr/share/vim/vim80/lang/zh_TW/LC_MESSAGES/vim.mo
/usr/share/vim/vim80/lang/zh_TW.UTF-8/LC_MESSAGES
/usr/share/vim/vim80/lang/zh_TW.UTF-8/LC_MESSAGES/vim.mo
```

```
/usr/share/vim/vim80/syntax/messages.vim
/usr/share/vim/vim80/syntax/msmessages.vim
/var/log/messages
```

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

```
[user@host ~]$ 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
```

3. SEARCHING FOR FILES IN REAL TIME

The **find** command locates files by performing a real-time search in the file-system hierarchy. It is slower than **locate**, but more accurate. It can also search for files based on criteria other than the file name, such as the permissions of the file, type of file, its size, or its modification time.

The **find** command looks at files in the file system using the user account that executed the search. 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** starts the search in the current directory and looks for matches in any subdirectory.

To search for files by file name, use the -**name *FILENAME*** option. With this option, **find** returns the path to files matching ***FILENAME*** exactly. For example, to search for files named **sshd_config** starting from the / directory, run the following command:

```
[root@host ~]# find / -name sshd_config
/etc/ssh/sshd_config
```

Wildcards are available to search for a file name and return all results that are a partial match. When using wildcards, it is important to quote the file name to look for to prevent the terminal from interpreting the wildcard.

In the following example, search for files starting in the / directory that end in **.txt**:

```
[root@host ~]# 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
...output omitted...
```

To search for files in the **/etc/** directory that contain the word, **pass**, anywhere in their names on host, run the following command:

```
[root@host ~]# find /etc -name '*pass*'
/etc/security/opasswd
/etc/pam.d/passwd
/etc/pam.d/password-auth
/etc/passwd-
/etc/passwd
```

```
/etc/authselect/password-auth
```

To perform a case-insensitive search for a given file name, use the **-iname** option, followed by the file name to search. To search files with case-insensitive text, **messages**, in their names in the / directory on host, run the following command:

```
[root@host ~]# find / -iname '*messages*'
...output omitted...
/usr/share/vim/vim80/lang/zh_CN.UTF-8/LC_MESSAGES
/usr/share/vim/vim80/lang/zh_CN.cp936/LC_MESSAGES
/usr/share/vim/vim80/lang/zh_TW/LC_MESSAGES
/usr/share/vim/vim80/lang/zh_TW.UTF-8/LC_MESSAGES
/usr/share/vim/vim80/syntax/messages.vim
/usr/share/vim/vim80/syntax/msmessages.vim
```

Searching Files Based on Ownership or Permission

The **find** command 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 user in the /home/user directory on host.

```
[user@host ~]$ find -user user
.
./.bash_logout
./.bash_profile
./.bashrc
./.bash_history
```

Search for files owned by the group user in the /home/user directory on host.

```
[user@host ~]$ find -group user
.
./.bash_logout
./.bash_profile
./.bashrc
./.bash_history
```

Search for files owned by user ID 1000 in the /home/user directory on host.

```
[user@host ~]$ find -uid 1000
.
./.bash_logout
./.bash_profile
./.bashrc./.bash_history
```

Search for files owned by group ID 1000 in the /home/user directory on host.

```
[user@host ~]$ find -gid 1000.
./.bash_logout
./.bash_profile
./.bashrc
./.bash_history
```

The **-user,** and **-group** options can be used together to search files where file owner and group owner are different. The example below list files that are both owned by user root and affiliated with group mail.

```
[root@host ~]# find / -user root -group mail
/var/spool/mail
...output omitted...
```

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 proceeded by / matches files that have at least one bit of user, group, or other for that permission set. A file with permission**s 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 matches 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@host ~]# find /home -perm 764
```

To match files for which the user has at least write and execute permissions, and the group has atleast write permissions, and others have at least read access:

```
[root@host ~]# find /home -perm -324
```

To match files for which the user has read permissions, or the group has at least read permissions, or others have at least write access:

```
[root@host ~]# find /home -perm /442
```

When used with / or -, a value of **0** works like a wildcard, since it means a permission of at least nothing.

To match any file in the **/home/user** directory for which others have at least read access on host,

run:

```
[user@host ~]$ find -perm -004
```

Find all files in the **/home/user** directory where other has write permissions on host.

[user@host ~]$ **find -perm -002**

Searching Files Based on Size

The find command can look up files that match a size specified with the -size option, followed by a numerical value and the unit. Use the following list as the units with the -size option:

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

The example below shows how to search for files with a size of 10 megabytes, rounded up.

[user@host ~]$ **find -size 10M**

To search the files with a size more than 10 gigabytes.

[user@host ~]$ **find -size +10G**

To list all files with a size less than 10 kilobytes.

[user@host ~]$ **find -size -10k**

Searching Files Based on Modification Time

The -**mmin** option, followed by the time in minutes, searches for all files that had their content changed at **n** minutes ago in the past. The file's timestamp is always rounded down. It also supports fractional values when used with ranges (**+n** and **-n**).

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

[root@host ~]# **find / -mmin 120**

The **+** modifier in front of the amount of minutes looks for all files in the / that have been modified more than n minutes ago. In this example, files that were modified more than 200 minutes ago are listed.

[root@host ~]# **find / -mmin +200**

The - modifier changes the search to look for all files in the / directory which have been changed less than n minutes ago. In this example, files that were modified less than 150 minutes ago are listed.

```
[root@host ~]# find / -mmin -150
```

Searching Files Based on File Type

The **-type** option in the **find** command limits the search scope to a given file type. Use the following list to pass the required flags to limit the scope of search:

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

Search for all directories in the **/etc** directory on host.

```
[root@host ~]# find /etc -type d

/etc

/etc/tmpfiles.d

/etc/systemd

/etc/systemd/system

/etc/systemd/system/getty.target.wants

...output omitted...
```

Search for all soft links on host.

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

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

```
[root@host ~]# find /dev -type b

/dev/vda1

/dev/vda
```

The **-links** option followed by a number looks for all files that have a certain hard link count. The number can be preceded by a + modifier to look for files with a count higher than the given hard link count. If the number is preceded with a - modifier, the search is limited to all files with a hard link count that is less than the given number.

Search for all regular files with more than one hard link on host:

```
[root@host ~]# find / -type f -links +1
```

Tasks: Locating Files on the System

1. Open an SSH session to localhost as student

```
[pllab@localhost ~]$ ssh student@localhost
[student@localhost ~]$
```

2. Use the locate command to search files on localhost.
   a. Even though the **locate** database is updated automatically every day, make sure that the database is up-to-date by manually starting an update on localhost. Use the **sudo updatedb** command to update the database used by the locate command.

   ```
   [student@localhost ~]$ sudo updatedb
   [student@localhost ~]$
   ```

   b. Locate the **logrotate.conf** configuration file.

   ```
   [student@localhost ~]$ locate logrotate.conf
   /etc/logrotate.conf
   /usr/share/man/man5/logrotate.conf.5.gz
   ```

   c. Locate the **networkmanager.conf** configuration file, ignoring case.

   ```
   [student@localhost ~]$ locate -i networkmanager.conf
   /etc/NetworkManager/NetworkManager.conf
   /etc/dbus-1/system.d/org.freedesktop.NetworkManager.conf
   /usr/share/man/man5/NetworkManager.conf.5.gz
   ```

3. Use the find command to perform real-time searches on localhost according to the following requirements:

- Search all files in the /var/lib directory that are owned by the chrony user.
- List all files in the /var directory that are owned by root and the group owner is mail.
- List all files in the /usr/bin directory that has a file size greater than 50 KB.
- Search all files in the /home/student directory that have not been changed in the last 120 minutes.
- List all the block device files in the /dev directory.

a. Use the find command to search all files in the /var/lib directory those are owned by the chrony user. Use the sudo command as the files inside the /var/lib directory are owned by root.

```
[student@localhost ~]$ sudo find /var/lib -user chrony
[sudo] password for student: student
/var/lib/chrony
/var/lib/chrony/drift
```

b. List all files in the **/var** directory that are owned by root and are affiliated with the mail group.

```
[student@localhost ~]$ sudo find /var -user root -group mail
/var/spool/mail
```

c. List all files in the /usr/bin directory with a file size greater than 50 KB.

```
[student@localhost ~]$ find /usr/bin -size +50k
/usr/bin/iconv
/usr/bin/locale
/usr/bin/localedef
/usr/bin/cmp
...output omitted...
```

d. Find all files in the /home/student directory that have not been changed in the last 120 minutes.

```
[student@localhost ~]$ find /home/student -mmin +120
/home/student/.bash_logout
/home/student/.bash_profile
/home/student/.bashrc
...output omitted...
```

e. List all block device files in the /dev directory.

```
[student@localhost ~]$ find /dev -type b
/dev/vdb
/dev/vda3
/dev/vda2
/dev/vda1
/dev/vda
```