

Filesystem Management

Michael C. Hackett
Computer Science Department

Community
College
of Philadelphia

Lecture Topics

- Managing Files and Directories
 - Creating Directories and Files
 - Moving Directories and Files
 - Renaming Directories and Files
 - Copying Directories and Files
 - Deleting Directories and Files
- Finding Files
- Linking Files
- File and Directory Permissions
 - Ownership
 - Managing Permissions
 - Default Permissions
 - Special Permissions
 - Access Control Lists
 - Filesystem Attributes

Creating Directories and Files

- The command used to **make** a **directory** is the **mkdir** command.

mkdir [argument]

- Argument is the directory you wish to create.

Creating Directories and Files

- The command **mkdir ExampleDirectory** would create a directory named ExampleDirectory.

```
[mhackett@localhost ~]$ ls
Desktop Documents Downloads Music Pictures Public Templates Videos
[mhackett@localhost ~]$ mkdir ExampleDirectory
[mhackett@localhost ~]$ ls
Desktop Documents Downloads ExampleDirectory Music Pictures Public Templates Videos
[mhackett@localhost ~]$ _
```

- To make a directory in a different location than the current working directory, provide the absolute path.
 - For example: **mkdir /var/log/ExampleDirectory**

Creating Directories and Files

- A command that is often used to create new files is the **touch** command.

touch [argument]

- Argument is the file you wish to create.
- The true purpose of the **touch** command is to update the last access date of a file.

Creating Directories and Files

- The command **touch NewFile.txt** would create a new (empty) file named NewFile.txt

```
[mhackett@localhost ~]$ ls
Desktop  Documents  Downloads  ExampleDirectory  Music  Pictures  Public  Templates  Videos
[mhackett@localhost ~]$ touch NewFile.txt
[mhackett@localhost ~]$ ls
Desktop  Downloads  Music  Pictures  Templates
Documents  ExampleDirectory  NewFile.txt  Public  Videos
[mhackett@localhost ~]$
```

- To create or touch a file in a different location than the current working directory, provide the absolute path.
 - For example: **touch /var/log/example.log**

Creating Directories and Files

- If the file already exists, the **touch** command will update the last access date of the file.

```
[mhackett@localhost ~]$ ls -la .bash_*
-rw-----. 1 mhackett mhackett 1769 Dec  2 00:54 .bash_history
-rw-r--r--. 1 mhackett mhackett  18 Aug  5 06:19 .bash_logout
-rw-r--r--. 1 mhackett mhackett 141 Aug  5 06:19 .bash_profile
[mhackett@localhost ~]$ date
Fri 06 Dec 2019 04:16:35 PM EST
[mhackett@localhost ~]$ touch .bash_profile
[mhackett@localhost ~]$ ls -la .bash_*
-rw-----. 1 mhackett mhackett 1769 Dec  2 00:54 .bash_history
-rw-r--r--. 1 mhackett mhackett  18 Aug  5 06:19 .bash_logout
-rw-r--r--. 1 mhackett mhackett 141 Dec  6 16:16 .bash_profile
[mhackett@localhost ~]$
```

- (The use of the **date** command was simply to display the system's current date and time prior to touching the existing file.)

Moving Directories and Files

- The command to **move** both directories and files is the **mv** command.

mv [options] *source destination*

- The source is the file or directory to move
- The destination is where the file or directory is to be moved to

Moving Directories and Files

- The command **mv ExampleDirectory Desktop** would move the ExampleDirectory directory into the Desktop directory.

```
[mhackett@localhost ~]$ ls
Desktop  Downloads  Music      Pictures  Templates
Documents ExampleDirectory NewFile.txt Public     Videos
[mhackett@localhost ~]$ mv ExampleDirectory Desktop
[mhackett@localhost ~]$ ls
Desktop Documents Downloads Music NewFile.txt Pictures Public Templates Videos
[mhackett@localhost ~]$ cd Desktop
[mhackett@localhost Desktop]$ ls
ExampleDirectory
[mhackett@localhost Desktop]$
```

Moving Directories and Files

- The command `mv ../NewFile.txt .` would move the NewFile.txt file (one directory up) into the current working directory.

```
[mhackett@localhost Desktop]$ ls
ExampleDirectory
[mhackett@localhost Desktop]$ mv ../NewFile.txt .
[mhackett@localhost Desktop]$ ls
ExampleDirectory NewFile.txt
[mhackett@localhost Desktop]$ _
```

Renaming Directories and Files

- The **mv** command is also used to rename a directory or file.

```
[mhackett@localhost Desktop]$ ls
ExampleDirectory  NewFile.txt
[mhackett@localhost Desktop]$ mv ExampleDirectory RenamedDirectory
[mhackett@localhost Desktop]$ mv NewFile.txt RenamedFile.txt
[mhackett@localhost Desktop]$ ls
RenamedDirectory  RenamedFile.txt
[mhackett@localhost Desktop]$
```

Copying Directories and Files

- The command to **copy** both directories and files is the **cp** command.

cp [options] *source destination*

- The source is the file or directory to copy
- The destination is where the file or directory is to be copied to

Copying Directories and Files

- The command **cp RenamedFile.txt CopiedFile.txt** would copy the RenamedFile.txt to a file (in the same working directory) named CopiedFile.txt

```
[mhackett@localhost Desktop]$ ls
RenamedDirectory RenamedFile.txt
[mhackett@localhost Desktop]$ cp RenamedFile.txt CopiedFile.txt
[mhackett@localhost Desktop]$ ls
CopiedFile.txt RenamedDirectory RenamedFile.txt
[mhackett@localhost Desktop]$
```

Copying Directories and Files

- The command
cp RenamedFile.txt RenamedDirectory/CopiedFile2.txt
would copy the RenamedFile.txt to a file named
CopiedFile.txt in the RenamedDirectory directory

```
[mhackett@localhost Desktop]$ ls
CopiedFile.txt  RenamedDirectory  RenamedFile.txt
[mhackett@localhost Desktop]$ cp RenamedFile.txt RenamedDirectory/CopiedFile2.txt
[mhackett@localhost Desktop]$ ls
CopiedFile.txt  RenamedDirectory  RenamedFile.txt
[mhackett@localhost Desktop]$ cd RenamedDirectory/
[mhackett@localhost RenamedDirectory]$ ls
CopiedFile2.txt
[mhackett@localhost RenamedDirectory]$
```

Copying Directories and Files

- The command `cp ../RenamedFile.txt .` would copy the RenamedFile.txt to a file to the current working directory.

```
[mhackett@localhost RenamedDirectory]$ ls
CopiedFile2.txt
[mhackett@localhost RenamedDirectory]$ cp ../RenamedFile.txt .
[mhackett@localhost RenamedDirectory]$ ls
CopiedFile2.txt  RenamedFile.txt
[mhackett@localhost RenamedDirectory]$
```

Copying Directories and Files

- The command `cp *.txt RenamedDirectory` would copy the any files that end with .txt at the current working directory into the RenamedDirectory directory.

```
[mhackett@localhost RenamedDirectory]$ cd ..  
[mhackett@localhost Desktop]$ ls  
CopiedFile.txt  RenamedDirectory  RenamedFile.txt  
[mhackett@localhost Desktop]$ cp *.txt RenamedDirectory  
[mhackett@localhost Desktop]$ ls RenamedDirectory  
CopiedFile2.txt  CopiedFile.txt  RenamedFile.txt  
[mhackett@localhost Desktop]$
```


Copying Directories and Files

- Notice the command did not prompt us before overwriting the existing RenamedFile.txt that already existed in the RenamedDirectory directory.

```
[mhackett@localhost RenamedDirectory]$ cd ..  
[mhackett@localhost Desktop]$ ls  
CopiedFile.txt  RenamedDirectory  RenamedFile.txt  
[mhackett@localhost Desktop]$ cp *.txt RenamedDirectory  
[mhackett@localhost Desktop]$ ls RenamedDirectory  
CopiedFile2.txt  CopiedFile.txt  RenamedFile.txt  
[mhackett@localhost Desktop]$
```

Copying Directories and Files

- To prompt before overwriting, use the **-i** option with the **cp** command.
 - **-i** is interactive mode

```
[mhackett@localhost Desktop]$ cp *.txt RenamedDirectory
[mhackett@localhost Desktop]$ ls
CopiedFile.txt  RenamedDirectory  RenamedFile.txt
[mhackett@localhost Desktop]$ cp -i *.txt RenamedDirectory
cp: overwrite 'RenamedDirectory/CopiedFile.txt'? n
cp: overwrite 'RenamedDirectory/RenamedFile.txt'? y
[mhackett@localhost Desktop]$ _
```

- The **-i** option also works with the **mv** command to prompt before overwriting existing files.

Copying Directories and Files

- When copying a *directory*, the **-r** option must be used.
 - **-r** (or **-R**) is the recursive option, which means all contents in the directory will be copied.

```
[mhackett@localhost Desktop]$ ls
CopiedFile.txt  RenamedDirectory  RenamedFile.txt
[mhackett@localhost Desktop]$ cp RenamedDirectory CopiedDirectory
cp: -r not specified; omitting directory 'RenamedDirectory'
[mhackett@localhost Desktop]$ cp -r RenamedDirectory CopiedDirectory
[mhackett@localhost Desktop]$ ls
CopiedDirectory  CopiedFile.txt  RenamedDirectory  RenamedFile.txt
[mhackett@localhost Desktop]$ ls RenamedDirectory
CopiedFile2.txt  CopiedFile.txt  RenamedFile.txt
[mhackett@localhost Desktop]$ ls CopiedDirectory
CopiedFile2.txt  CopiedFile.txt  RenamedFile.txt
[mhackett@localhost Desktop]$
```

Deleting Directories and Files

- The command to **remove** files is the **rm** command.

rm [options] [arguments]

- The **rm** command (with the **-r** option) is used to remove non-empty directories.

rm -r [arguments]

- The **rmdir** command is used to remove empty directories.

rmdir [options] [arguments]

Deleting Directories and Files

- The command `rm RenamedFile.txt` would delete the RenamedFile.txt file in the current working directory.

```
[mhackett@localhost Desktop]$ ls
CopiedDirectory CopiedFile.txt RenamedDirectory RenamedFile.txt
[mhackett@localhost Desktop]$ rm RenamedFile.txt
[mhackett@localhost Desktop]$ ls
CopiedDirectory CopiedFile.txt RenamedDirectory
[mhackett@localhost Desktop]$ _
```

Deleting Directories and Files

- The command `rmdir RenamedDirectory` would *attempt* to delete the `RenamedDirectory` directory in the current working directory.
 - This will not work for non-empty directories.

```
[mhackett@localhost Desktop]$ ls
CopiedDirectory CopiedFile.txt RenamedDirectory
[mhackett@localhost Desktop]$ rmdir RenamedDirectory
rmdir: failed to remove 'RenamedDirectory': Directory not empty
[mhackett@localhost Desktop]$ _
```

Deleting Directories and Files

- The `rm -r` command and option is used to remove non-empty directories.

```
[mhackett@localhost Desktop]$ ls
CopiedDirectory CopiedFile.txt RenamedDirectory
[mhackett@localhost Desktop]$ rmdir RenamedDirectory
rmdir: failed to remove 'RenamedDirectory': Directory not empty
[mhackett@localhost Desktop]$ rm -r RenamedDirectory
[mhackett@localhost Desktop]$ ls
CopiedDirectory CopiedFile.txt
[mhackett@localhost Desktop]$
```

Finding Files

- When searching for files, there are different commands for different types of searches.
- To search the *entire* filesystem for a file based on its name, the locate command is used.

locate [argument]

- The argument is the name (or partial name) of the file.

Finding Files

- The command **locate fstab** would search the entire filesystem for files that contain “fstab” in its name and display the results.

```
[mhackett@localhost Desktop]$ locate fstab
/etc/fstab
/usr/lib/dracut/modules.d/95fstab-sys
/usr/lib/dracut/modules.d/95fstab-sys/module-setup.sh
/usr/lib/dracut/modules.d/95fstab-sys/mount-sys.sh
/usr/lib/systemd/system-generators/systemd-fstab-generator
/usr/share/augeas/lenses/dist/fstab.aug
/usr/share/augeas/lenses/dist/vfstab.aug
/usr/share/man/man5/fstab.5.gz
/usr/share/man/man8/systemd-fstab-generator.8.gz
[mhackett@localhost Desktop]$ _
```

Finding Files

- The **locate** command maintains a database of files in the filesystem.
 - Actual database file is `/var/lib/mlocate/mlocate.db`
- As files are added and removed, the database needs to be updated.
 - This is done automatically once a day.
 - To manually update the database, run the **updatedb** command as the root user.

Finding Files

```
[mhackett@localhost ~]$ su -  
Password:  
[root@localhost ~]# locate CopiedFile  
/home/mhackett/Desktop/CopiedFile.txt  
[root@localhost ~]# updatedb  
[root@localhost ~]# locate CopiedFile  
/home/mhackett/Desktop/CopiedFile.txt  
/home/mhackett/Desktop/CopiedDirectory/CopiedFile.txt  
/home/mhackett/Desktop/CopiedDirectory/CopiedFile2.txt  
[root@localhost ~]# _
```

Finding Files

- Another command for searching for files is the **find** command.
 - find start [options] argument**
 - The start is the directory to begin the search.
 - The options specify the type of criteria.
 - The argument is the criteria for the search.
- It is slower than **locate**, but is more versatile.
 - Slower because it doesn't maintain a database of files like **locate** does

Finding Files

- The command `find /etc -name "hosts"` would search the entire etc directory for files that contain “hosts” in its name and display the results.

```
[root@localhost ~]# find /etc -name "hosts"
/etc/hosts
/etc/avahi/hosts
[root@localhost ~]#
```

Finding Files

- The command `find /etc -name "host*"` would search the entire etc directory for files that begin with “host” followed by any characters and display the results.

```
[root@localhost ~]# find /etc -name "host*"
/etc/host.conf
/etc/hosts
/etc/avahi/hosts
/etc/hostname
[root@localhost ~]# _
```

Finding Files

- The command `find /var -size +25M` would search the entire var directory for files larger than 25 megabytes.

```
[root@localhost ~]# find /var -size +25M
/var/lib/rpm/Packages
/var/cache/PackageKit/31/metadata/fedora-31-x86_64/repodata/5dd866cf6ce0e21f428b6c5bf4eabee65719b72c8cc279393125e79b34fdae31-primary.xml.zck
/var/cache/PackageKit/31/metadata/fedora-31-x86_64/repodata/d601f9ef02bca6948263031733c69a1dbdc8ad11750d4e8dbd91256d775ecaf5-filelists.xml.zck
/var/cache/PackageKit/31/hawkey/fedora-filenames.solvx
/var/cache/dnf/fedora-3589ee8a7ee1691d/repodata/5dd866cf6ce0e21f428b6c5bf4eabee65719b72c8cc279393125e79b34fdae31-primary.xml.zck
/var/cache/dnf/fedora-3589ee8a7ee1691d/repodata/d601f9ef02bca6948263031733c69a1dbdc8ad11750d4e8dbd91256d775ecaf5-filelists.xml.zck
/var/cache/dnf/fedora-filenames.solvx
[root@localhost ~]# _
```

- See the manual for `find` (enter `man find`) for additional criteria.

Finding Files

- The **which** command searches locations specified in the user's PATH environment variable.

which argument

- The argument is the name of the file or program.

Finding Files

- The command **which cp** would search only the user's PATH for files or programs named "cp"
 - The \$PATH is displayed to illustrate where the **which** command will search.

```
[mhackett@localhost ~]$ echo $PATH
/home/mhackett/.local/bin:/home/mhackett/bin:/usr/share/Modules/bin:/usr/local/bin:/usr/local/sbin:/usr/sbin
[mhackett@localhost ~]$ which cp
/usr/bin/cp
[mhackett@localhost ~]$ _
```

Finding Files

- A command similar to the **which** command is the **type** command.

type argument

- The argument is the name of the file or program.
- If there were more than one result of the **which** command, the **type** command would only display the first result.

Finding Files

- The **whereis** command displays the location of the file or program (searches only the directories in the user's PATH) and any associated manual files.

whereis argument

- The argument is the name of the file or program.

Finding Files

- The command **whereis cp** would search only the user's PATH for files or programs named "cp" and its associated manual files.

```
[mhackett@localhost ~]$ whereis cp
cp: /usr/bin/cp /usr/share/man/man1/cp.1.gz /usr/share/man/man1p/cp.1p.gz
[mhackett@localhost ~]$
```

Linking Files

- The filesystem, structurally, has three main sections
 - The **superblock** which contains (among other things) the number of inodes and size of data blocks.
 - The **inode table** contains a table of information **nodes** which each describe a file or directory in the filesystem. Each inode has a unique identification number.
 - inodes store information about their files and directories like its size, permissions, ownership, etc.
 - The **data blocks** (or **allocation units**) contain the actual data of files.
- When a file or directory is deleted, only its inode is deleted.

Linking Files

- A file is **hard linked** when it shares the same inode as another file.
 - They share the same inode number and data blocks.
- Hard linked files must all reside in the same filesystem.

Linking Files

- A file is **soft linked** when it simply points another file.
 - The proper term is **symbolically linked** or a “**symlink**”
 - They do not share the same inode number and data blocks.
- Soft linked files do not need to all reside in the same filesystem.

Linking Files

- The **ln** command is used to create both hard **links** and **symlinks**

ln [options] source link

- To create a hard link, no options are needed:

ln origFile.txt origFileLink.txt

- To create a symlink, use the **-s** option:

ln -s origFile.txt origFileLink.txt

Linking Files

- To include inode numbers in a listing, use the **-i** option with the **ls** command:

```
[root@localhost ~]# cp /etc/inittab ./OrigFile.txt
[root@localhost ~]# cp OrigFile.txt OrigFile2.txt
[root@localhost ~]# ls -l
total 12
-rw----- . 1 root root 1333 Nov 30 13:58 anaconda-ks.cfg
-rw-r--r-- . 1 root root 490 Dec 7 14:49 OrigFile2.txt
-rw-r--r-- . 1 root root 490 Dec 7 14:49 OrigFile.txt
[root@localhost ~]# ls -li
total 12
16777346 -rw----- . 1 root root 1333 Nov 30 13:58 anaconda-ks.cfg
19785818 -rw-r--r-- . 1 root root 490 Dec 7 14:49 OrigFile2.txt
16777355 -rw-r--r-- . 1 root root 490 Dec 7 14:49 OrigFile.txt
[root@localhost ~]# _
```

Linking Files

- Examples of creating hard links and symlinks:

```
[root@localhost ~]# ln OrigFile.txt HardLink
[root@localhost ~]# ln -s OrigFile2.txt SoftLink
[root@localhost ~]# ls -li
total 16
16777346 -rw-----. 1 root root 1333 Nov 30 13:58 anaconda-ks.cfg
16777355 -rw-r--r--. 2 root root 490 Dec 7 14:49 HardLink
19785818 -rw-r--r--. 1 root root 490 Dec 7 14:49 OrigFile2.txt
16777355 -rw-r--r--. 2 root root 490 Dec 7 14:49 OrigFile.txt
19785826 lrwxrwxrwx. 1 root root 13 Dec 7 14:52 SoftLink -> OrigFile2.txt
[root@localhost ~]# _
```

Linking Files

- Notice the hard linked file shares the same inode number:

```
[root@localhost ~]# ln OrigFile.txt HardLink
[root@localhost ~]# ln -s OrigFile2.txt SoftLink
[root@localhost ~]# ls -li
total 16
16777346 -rw-----. 1 root root 1333 Nov 30 13:58 anaconda-ks.cfg
16777355 -rw-r--r--. 2 root root 490 Dec 7 14:49 HardLink
19785818 -rw-r--r--. 1 root root 490 Dec 7 14:49 OrigFile2.txt
16777355 -rw-r--r--. 2 root root 490 Dec 7 14:49 OrigFile.txt
19785826 lrwxrwxrwx. 1 root root 13 Dec 7 14:52 SoftLink -> OrigFile2.txt
[root@localhost ~]# _
```

Linking Files

- Notice the symlinks do not share the same inode number:

```
[root@localhost ~]# ln OrigFile.txt HardLink
[root@localhost ~]# ln -s OrigFile2.txt SoftLink
[root@localhost ~]# ls -li
total 16
16777346 -rw----- . 1 root root 1333 Nov 30 13:58 anaconda-ks.cfg
16777355 -rw-r--r-- . 2 root root 490 Dec 7 14:49 HardLink
19785818 -rw-r--r-- . 1 root root 490 Dec 7 14:49 OrigFile2.txt
16777355 -rw-r--r-- . 2 root root 490 Dec 7 14:49 OrigFile.txt
19785826 lrwxrwxrwx . 1 root root 13 Dec 7 14:52 SoftLink -> OrigFile2.txt
[root@localhost ~]# _
```

- Just before the file's mode (discussed shortly and mentioned in a previous lecture), an l indicates a symlink

Linking Files

- Also note the link counts for each file:
 - Symlinks do not increase the link count

```
[root@localhost ~]# ln OrigFile.txt HardLink
[root@localhost ~]# ln -s OrigFile2.txt SoftLink
[root@localhost ~]# ls -li
total 16
16777346 -rw----- . 1 root root 1333 Nov 30 13:58 anaconda-ks.cfg
16777355 -rw-r--r-- . 2 root root 490 Dec 7 14:49 HardLink
19785818 -rw-r--r-- . 1 root root 490 Dec 7 14:49 OrigFile2.txt
16777355 -rw-r--r-- . 2 root root 490 Dec 7 14:49 OrigFile.txt
19785826 lrwxrwxrwx . 1 root root 13 Dec 7 14:52 SoftLink -> OrigFile2.txt
[root@localhost ~]#
```

Linking Files

- Deleting a link will update the link count:

```
[root@localhost ~]# rm HardLink
rm: remove regular file 'HardLink'? y
[root@localhost ~]# ls -li
total 12
16777346 -rw----- . 1 root root 1333 Nov 30 13:58 anaconda-ks.cfg
19785818 -rw-r--r-- . 1 root root 490 Dec 7 14:49 OrigFile2.txt
16777355 -rw-r--r-- . 1 root root 490 Dec 7 14:49 OrigFile.txt
19785826 lrwxrwxrwx . 1 root root 13 Dec 7 14:52 SoftLink -> OrigFile2.txt
[root@localhost ~]# _
```

Linking Files

- Another important distinction:
 - Directories cannot be hard linked
 - Directories can be symbolically linked

```
[root@localhost ~]# ln /etc etclink
ln: /etc: hard link not allowed for directory
[root@localhost ~]# ln -s /etc etclink
[root@localhost ~]# ls -li
total 12
16777346 -rw-----. 1 root root 1333 Nov 30 13:58 anaconda-ks.cfg
19785827 lrwxrwxrwx. 1 root root   4 Dec  7 15:41 etclink -> /etc
19785818 -rw-r--r--. 1 root root  490 Dec  7 14:49 OrigFile2.txt
16777355 -rw-r--r--. 1 root root  490 Dec  7 14:49 OrigFile.txt
19785826 lrwxrwxrwx. 1 root root  13 Dec  7 14:52 SoftLink -> OrigFile2.txt
[root@localhost ~]# _
```

- Use the unlink command to delete a directory symlink
unlink *directory*

File and Directory Permissions

- Every user is identified by their username and a group membership.
 - Users can be members of multiple groups.
- Access to resources like files and directories depends on the resource's **permissions**.
 - Access is granted based on the resource's owner and group ownership.

Ownership

- When a user creates files and directories, the file or directory is created with
 - The user as the owner
 - The user's primary group as the group owner
- The **groups** command will list the current user's group membership.
 - The first group listed is the user's primary group
 - Most users will have a primary group that is identical to their username

Ownership

- Currently, this user account is only a member of one group.
 - We'll see how to add a user to additional groups in a later lecture.

```
[mhackett@localhost ~]$ groups  
mhackett
```

Ownership

- When creating a new file (or directory) we see that the default ownership is the user that created it and their primary group.

```
[mhackett@localhost ~]$ touch PermFile.txt
[mhackett@localhost ~]$ ls -l PermFile.txt
-rw-rw-r--. 1 mhackett mhackett 0 Dec  7 11:35 PermFile.txt
[mhackett@localhost ~]$
```

Ownership

- The command to **change ownership** of a file or directory is the **chown** command.

chown [options] *username* [files or directories]

- Only the root user or current owner* can change the resource's ownership
 - *(Unless the current owner does not have access/permission to use the chown command; discussed later)

Ownership

- Changing ownership of PermFile.txt from mhackett to the root user:

```
[mhackett@localhost ~]$ ls -l PermFile.txt
-rw-rw-r--. 1 mhackett mhackett 0 Dec  7 11:35 PermFile.txt
[mhackett@localhost ~]$ chown root PermFile.txt
chown: changing ownership of 'PermFile.txt': Operation not permitted
[mhackett@localhost ~]$ su - root
Password:
[root@localhost ~]# chown root /home/mhackett/PermFile.txt
[root@localhost ~]# exit
logout
[mhackett@localhost ~]$ ls -l PermFile.txt
-rw-rw-r--. 1 root mhackett 0 Dec  7 11:35 PermFile.txt
[mhackett@localhost ~]$ _
```

Ownership

- To change a directory's owner, the syntax is the same:
chown *newowner directory*
- To change a directory's owner and change the owner of everything in it, use the -R (recursive) option:
chown -R *newowner directory*

Ownership

- The command to **change group** ownership of a file or directory is the **chgrp** command.
chgrp [options] *username* [files or directories]
- Only the root user, current owner*, or a group member* can change the resource's group ownership
 - *Can only change the group to a group that the owner or group member belongs to

Ownership

- Changing group ownership of PermFile.txt from the mhackett to the root group:
 - (mhackett is not a member of the root group)

```
[mhackett@localhost ~]$ chgrp root PermFile.txt
chgrp: changing group of 'PermFile.txt': Operation not permitted
[mhackett@localhost ~]$ su - root
Password:
[root@localhost ~]# chgrp root /home/mhackett/PermFile.txt
[root@localhost ~]# exit
logout
[mhackett@localhost ~]$ ls -l PermFile.txt
-rw-rw-r--. 1 root root 0 Dec  7 11:35 PermFile.txt
[mhackett@localhost ~]$
```


Ownership

- To change a directory's group ownership, the syntax is the same:

chgrp newgroup directory

- To change a directory's group ownership and change the group ownership of everything in it, use the **-R** (recursive) option:

chgrp -R newgroup directory

Ownership

- The **chgrp** command will only work with existing groups.
- Creating new groups is a topic covered in a later lecture.

Ownership

- The **chown** command can be used to change both owner and group at the same time.

chown [options] *username:group* [files or directories]

```
[root@localhost ~]# ls -l /home/mhackett/PermFile.txt
-rw-rw-r--. 1 root root 0 Dec  7 11:35 /home/mhackett/PermFile.txt
[root@localhost ~]# chown mhackett:mhackett /home/mhackett/PermFile.txt
[root@localhost ~]# ls -l /home/mhackett/PermFile.txt
-rw-rw-r--. 1 mhackett mhackett 0 Dec  7 11:35 /home/mhackett/PermFile.txt
[root@localhost ~]#
```

Managing Permissions

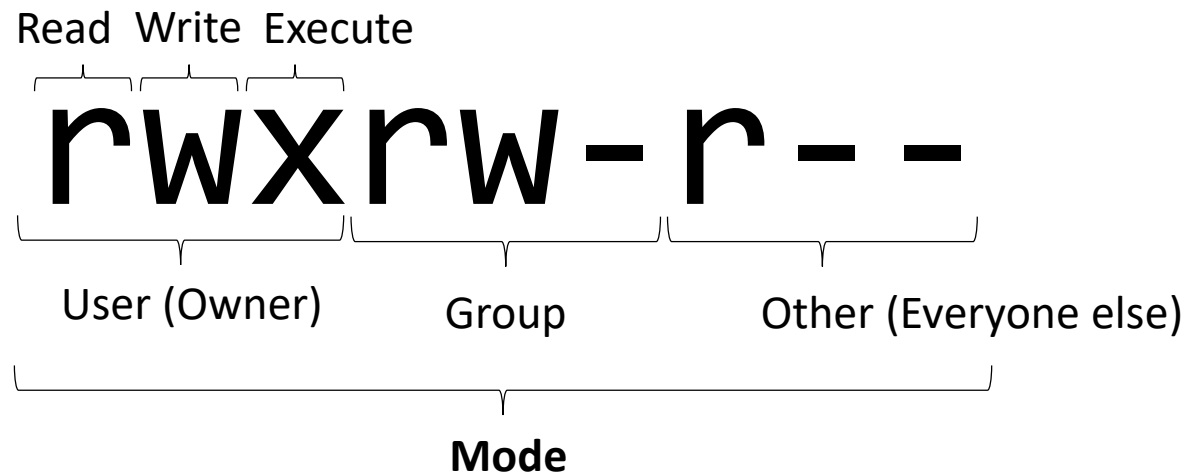
- Every file and directory has a set of permissions called its **mode**
- The mode is broken down into three main groups:
 - User (owner) permissions
 - Group permissions
 - Other (everyone else) permissions
- Each group is given some combination (or none) of the following permissions:
 - Read Permission
 - Write Permission
 - Execute Permission

Managing Permissions

- Read Permission
 - For files: Allows opening and reading the file
 - For directories: Allows listing the contents of the directory (requires execute permission)
- Write Permission
 - For files: Allows opening, reading, and editing the file
 - For directories: Allows adding and removing to/from the directory (requires execute permission)
- Execute Permission
 - For files: Allows executing the file (if it is a program or script)
 - For directories: Allows entering the directory and to work with its contents.

Managing Permissions

- The structure of a file or directory's mode:
 - - means the permission is not granted



Managing Permissions

- A file or directory with these permissions would permit the following:

`rwxrw-r--`

The permissions `rwxrw-r--` are grouped into three categories by brackets below them:

- User (Owner)**: `rwx`
- Group**: `rw-`
- Other (Everyone else)**: `r--`

- User: Read, Write, and Execute Permission
- Group: Read and Write Permission
- Other: Read Permission

Managing Permissions

- The modes of files and directories can be viewed with the **ls -l** command:

```
[mhackett@localhost ~]$ ls -l
total 0
drwxr-xr-x. 3 mhackett mhackett 51 Dec  7 00:34 Desktop
drwxr-xr-x. 2 mhackett mhackett  6 Nov 30 14:19 Documents
drwxr-xr-x. 2 mhackett mhackett  6 Nov 30 14:19 Downloads
drwxr-xr-x. 2 mhackett mhackett  6 Nov 30 14:19 Music
-rw-rw-r--. 1 mhackett mhackett  0 Dec  7 11:35 PermFile.txt
drwxr-xr-x. 2 mhackett mhackett 100 Nov 30 14:34 Pictures
drwxr-xr-x. 2 mhackett mhackett  6 Nov 30 14:19 Public
drwxr-xr-x. 2 mhackett mhackett  6 Nov 30 14:19 Templates
drwxr-xr-x. 2 mhackett mhackett  6 Nov 30 14:19 Videos
[mhackett@localhost ~]$ _
```


Managing Permissions

- To **change** a file or directory's **mode**, the `chmod` command is used.

`chmod [options] file/directory`

- The options are the criteria for the mode.

Managing Permissions

- Criteria categories:
 - **u** (User)
 - **g** (Group)
 - **o** (Other)
 - **a** (All- User, Group, and Other)
- Criteria operations:
 - **+** (Adds a permission)
 - **-** (Removes a permission)
 - **=** (Sets permissions exactly to)
- Criteria permissions:
 - **r** (Read)
 - **w** (Write)
 - **x** (Execute)

Managing Permissions

- Examples:

chmod g+x somefile.txt

- Would add executable permission to the group

chmod u+w,o+x somefile.txt

- Would add write permission to the owner and executable permission to others

chmod o-w somefile.txt

- Would remove write permission from the group

chmod u=rwx,g=rw,o-x somefile.txt

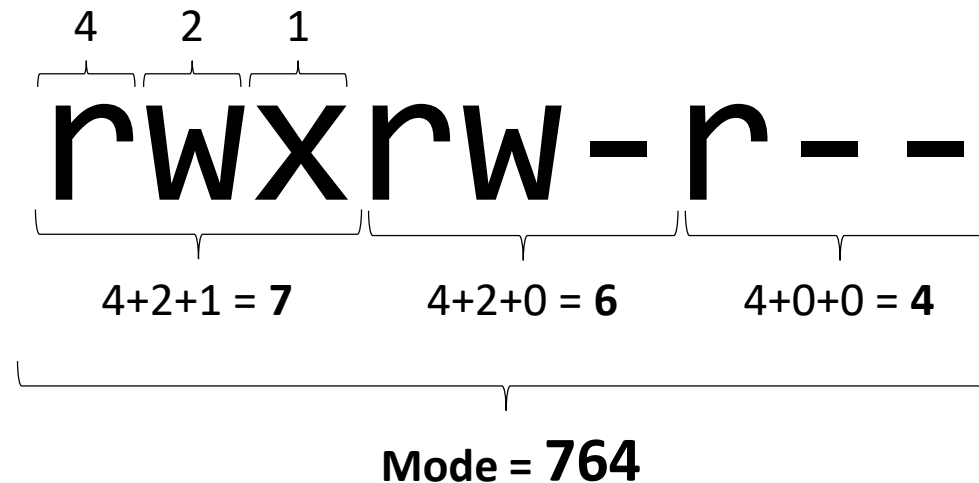
- Would set read, write, and execute permission to the owner, read and write permission to the group, and remove execute permissions from others

Managing Permissions

```
[mhackett@localhost ~]$ ls -l PermFile.txt
-rw-rw-r--. 1 mhackett mhackett 0 Dec  7 11:35 PermFile.txt
[mhackett@localhost ~]$ chmod o+x PermFile.txt
[mhackett@localhost ~]$ ls -l PermFile.txt
-rw-rw-r-x. 1 mhackett mhackett 0 Dec  7 11:35 PermFile.txt
[mhackett@localhost ~]$ chmod g-w PermFile.txt
[mhackett@localhost ~]$ ls -l PermFile.txt
-rw-r--r-x. 1 mhackett mhackett 0 Dec  7 11:35 PermFile.txt
[mhackett@localhost ~]$ chmod u=rwx,g=wx,o-x PermFile.txt
[mhackett@localhost ~]$ ls -l PermFile.txt
-rwx-wxr--. 1 mhackett mhackett 0 Dec  7 11:35 PermFile.txt
[mhackett@localhost ~]$ _
```

Managing Permissions

- File permissions can also be set using numeric (octal) values.
 - Read = 4
 - Write = 2
 - Execute = 1



Managing Permissions

- Eight possible combinations:

`rwX` $4+2+1 = \mathbf{7}$

`rw-` $4+2+0 = \mathbf{6}$

`r-X` $4+0+1 = \mathbf{5}$

`r--` $4+0+0 = \mathbf{4}$

`-wX` $0+2+1 = \mathbf{3}$

`-w-` $0+2+0 = \mathbf{2}$

`--X` $0+0+1 = \mathbf{1}$

`---` $0+0+0 = \mathbf{0}$

Managing Permissions

- Examples:

chmod 777 somefile.txt

- Would result in a mode of `rw-rw-rw-`

chmod 742 somefile.txt

- Would result in a mode of `rw-r--w-`

chmod 000 somefile.txt

- Would result in a mode of `-----`

chmod 640 somefile.txt

- Would result in a mode of `rw-r----`

Managing Permissions

```
[mhackett@localhost ~]$ ls -l PermFile.txt
-rwx-wxr--. 1 mhackett mhackett 0 Dec  7 11:35 PermFile.txt
[mhackett@localhost ~]$ chmod 777 PermFile.txt
[mhackett@localhost ~]$ ls -l PermFile.txt
-rwxrwxrwx. 1 mhackett mhackett 0 Dec  7 11:35 PermFile.txt
[mhackett@localhost ~]$ chmod 640 PermFile.txt
[mhackett@localhost ~]$ ls -l PermFile.txt
-rw-r-----. 1 mhackett mhackett 0 Dec  7 11:35 PermFile.txt
[mhackett@localhost ~]$ _
```


Default Permissions

- By default, files are created with the following permissions:
rw-rw-rw-
- By default, directories are created with the following permissions:
rwxrwxrwx
- A special variable on the system called the **user mask (umask)** immediately takes away permissions from newly created files and directories.

Default Permissions

- A common umask is 022
 - This specifies nothing is taken away from the owner, write permission is removed from the group, and write permission is removed from everyone else

Default Permissions

- New Files (umask is 022):

| | | | |
|------------------|-----------|-----------|-----------|
| rw-rw-rw- | 6 | 6 | 6 |
| | <u>-0</u> | <u>-2</u> | <u>-2</u> |
| rw-r--r-- | 6 | 4 | 4 |

- New Directories (umask is 002):

| | | | |
|------------------|-----------|-----------|-----------|
| rw-rw-rwx | 7 | 7 | 7 |
| | <u>-0</u> | <u>-2</u> | <u>-2</u> |
| rw-r-xr-x | 7 | 5 | 5 |

Default Permissions

- To view the system's current user mask, enter the **umask** command.

```
[mhackett@localhost ~]$ umask  
0002  
[mhackett@localhost ~]$
```

- On this system, write permission for other users is the only thing removed from new files and directories.
- The last slides showed only three numbers for the user mask
 - The first digit is used for a special type of permission discussed shortly.

Default Permissions

- Changing the user mask:

```
[mhackett@localhost ~]$ umask
0002
[mhackett@localhost ~]$ touch masktest.txt
[mhackett@localhost ~]$ ls -l masktest.txt
-rw-rw-r--. 1 mhackett mhackett 0 Dec  7 13:31 masktest.txt
[mhackett@localhost ~]$ umask 022
[mhackett@localhost ~]$ touch masktest2.txt
[mhackett@localhost ~]$ ls -l masktest*
-rw-r--r--. 1 mhackett mhackett 0 Dec  7 13:32 masktest2.txt
-rw-rw-r--. 1 mhackett mhackett 0 Dec  7 13:32 masktest.txt
[mhackett@localhost ~]$
```

Special Permissions

- Read, write, and execute are the regular permissions for files.
- There are three more special permissions:
 - SUID (Set User ID)
 - SGID (Set Group ID)
 - Sticky bit

Special Permissions

- When SUID is set for a file, the user temporarily becomes the file's owner when the file is executed.
 - No effect on directories
- When SGID is set for a file, the user temporarily becomes a member of the file's group ownership when the file is executed.
 - For directories, new files created will have the user as the owner, but will have the *directory's* group ownership

Special Permissions

- When used on a directory, the sticky bit allows users to write to directory but only delete files they own.
- The sticky bit, in the past, was used to lock files in memory.
 - Today, the sticky bit is only used for directories.

Special Permissions

- Special permissions are seen in file or directory's mode:

The diagram illustrates the mapping of special permissions from octal mode to symbolic mode. It consists of two rows of text. The top row is 'rwxrwxrwx' in a large, bold, monospace font. Below this row, there are three vertical arrows pointing downwards. The first arrow is positioned under the first 'r' and is labeled 'SUID' to its left. The second arrow is positioned under the second 'x' and is labeled 'SGID' to its left. The third arrow is positioned under the last 'x' and is labeled 'Sticky bit' to its left. The bottom row is 'rwsrwsrwt' in the same large, bold, monospace font. The 'SUID' arrow points from the first 'r' to the first 's'. The 'SGID' arrow points from the second 'x' to the second 's'. The 'Sticky bit' arrow points from the last 'x' to the last 't'.

rwxrwxrwx

SUID SGID Sticky bit

rwsrwsrwt

Special Permissions

- If execute permission is not enabled, they appear as capital letters:

The diagram illustrates the conversion of lowercase permission letters to uppercase when the execute permission is not enabled. It shows three columns of permissions: 'rw-', 'rw-', and 'rw-'. Arrows labeled 'SUID', 'SGID', and 'Sticky bit' point from the lowercase 'w' in each column to the corresponding uppercase letter 'S' or 'T' in the resulting permission string 'rwSrwsrwT'.

rw- rw- rw-

SUID SGID Sticky bit

rwSrwsrwT

Special Permissions

- Special permissions are set using numeric (octal) values.
 - SUID = 4
 - SGID = 2
 - Sticky bit = 1

Special Permissions

- Eight possible combinations:
(SUID SGID Stickybit)

| | |
|-----|-------------|
| sst | $4+2+1 = 7$ |
| ss- | $4+2+0 = 6$ |
| s-t | $4+0+1 = 5$ |
| s-- | $4+0+0 = 4$ |
| -st | $0+2+1 = 3$ |
| -s- | $0+2+0 = 2$ |
| --t | $0+0+1 = 1$ |
| --- | $0+0+0 = 0$ |

Special Permissions

- Set by providing a fourth number to the **chmod** command.
 - The special permissions is the first number

chmod 641 somedirectory

- Results in rw-r----x

chmod 5641 somedirectory

- 5 = SUID and Sticky bit are enabled
- Results in rwSr---t
 - S because owner execute permission is disabled
 - t because all execute permission is enabled

Special Permissions

```
[root@localhost ~]# chmod 666 OrigFile.txt
[root@localhost ~]# ls -l OrigFile.txt
-rw-rw-rw-. 1 root root 490 Dec  7 14:49 OrigFile.txt
[root@localhost ~]# chmod 7666 OrigFile.txt
[root@localhost ~]# ls -l OrigFile.txt
-rwsrwSrwt. 1 root root 490 Dec  7 14:49 OrigFile.txt
[root@localhost ~]# chmod 0777 OrigFile.txt
[root@localhost ~]# ls -l OrigFile.txt
-rwxrwxrwx. 1 root root 490 Dec  7 14:49 OrigFile.txt
[root@localhost ~]# chmod 7777 OrigFile.txt
[root@localhost ~]# ls -l OrigFile.txt
-rwsrwsrwt. 1 root root 490 Dec  7 14:49 OrigFile.txt
[root@localhost ~]# _
```

Access Control Lists

- An **Access Control List (ACL)** is a list of additional users or groups that you can assign permissions to for a file or directory.
- ACLs allow you to specify permissions to a file or directory for one user or group (in addition to the owner and owning group) instead of an entire group or everyone else.

Access Control Lists

- To update or **set** a **file's access control list**, the **setfacl** command is used.

- Used for files and directories

setfacl [options] [criteria] file/directory

- The **-m** option is used to modify a file or directory's ACL
- The syntax for user criteria is **u:name:permissions**
- The syntax for group criteria is **g:name:permissions**

Access Control Lists

- The command **setfacl -m u:mhackett:r-- somefile.txt** would give the user mhackett read permission (but not write and execute) to somefile.txt
 - Note the **+** symbol that indicates the presence of additional permissions

```
[root@localhost ~]# touch somefile.txt
[root@localhost ~]# chmod 740 somefile.txt
[root@localhost ~]# ls -l somefile.txt
-rwxr-----. 1 root root 0 Dec  7 16:36 somefile.txt
[root@localhost ~]# setfacl -m u:mhackett:r-- somefile.txt
[root@localhost ~]# ls -l somefile.txt
-rwxr-----+ 1 root root 0 Dec  7 16:36 somefile.txt
[root@localhost ~]#
```

Access Control Lists

- To view or **get** a file's **access control list**, the **getfacl** command is used.
 - Used for files and directories
- getfacl file/directory**

```
[root@localhost ~]# ls -l somefile.txt
-rwxr-----+ 1 root root 0 Dec  7 16:36 somefile.txt
[root@localhost ~]# getfacl somefile.txt
# file: somefile.txt
# owner: root
# group: root
user::rwx
user:mhackett:r--
group::r--
mask::r--
other::---
```

[root@localhost ~]# _

Access Control Lists

- To remove all additional permissions in the ACL, use the **-b** option with the **setfacl** command.

```
[root@localhost ~]# setfacl -b somefile.txt
[root@localhost ~]# ls -l somefile.txt
-rwxr-----. 1 root root 0 Dec  7 16:36 somefile.txt
[root@localhost ~]# getfacl somefile.txt
# file: somefile.txt
# owner: root
# group: root
user::rwx
group::r--
other::---

[root@localhost ~]# _
```

Filesystem Attributes

- Files and directories can have attributes that work outside of the standard permissions.
- The attributes are filesystem dependent.
 - Types of filesystems were briefly discussed in a previous lecture.
 - The next lecture goes more in depth on types of filesystems.

Filesystem Attributes

- To list a file or directory's **attributes**, the **lsattr** command is used.

lsattr file/directory

- The example below shows this file has no attributes set.

```
[root@localhost ~]# lsattr somefile.txt
----- somefile.txt
```

Filesystem Attributes

- To **change** a file or directory's **attributes**, the **chattr** command is used.
`chattr [criteria] file/directory`
- We won't cover every attribute, just the **immutable** attribute.
 - This prevents a file or directory from being changed or deleted by anyone.
 - Use **man chattr** for details about additional attributes in the chattr user manual.

Filesystem Attributes

- The command **chattr +i somefile.txt** would add the immutable attribute to somefile.txt.
 - Not able to be modified, moved, or deleted

```
[root@localhost ~]# lsattr somefile.txt
----- somefile.txt
[root@localhost ~]# chattr +i somefile.txt
[root@localhost ~]# lsattr somefile.txt
----i----- somefile.txt
[root@localhost ~]# rm somefile.txt
rm: remove regular empty file 'somefile.txt'? y
rm: cannot remove 'somefile.txt': Operation not permitted
[root@localhost ~]# mv somefile.txt /etc
mv: cannot move 'somefile.txt' to '/etc/somefile.txt': Operation not permitted
[root@localhost ~]#
```

Filesystem Attributes

- The command **chattr -i somefile.txt** would remove the immutable attribute to somefile.txt.
 - Able to be modified, moved, or deleted again

```
[root@localhost ~]# lsattr somefile.txt
----i----- somefile.txt
[root@localhost ~]# chattr -i somefile.txt
[root@localhost ~]# lsattr somefile.txt
----- somefile.txt
[root@localhost ~]# rm somefile.txt
rm: remove regular empty file 'somefile.txt'? y
[root@localhost ~]# _
```