Linux-Basic-Commands

Working on the Linux command line can be intimidating, at first. However, like anything, with a little practice, you will become proficient. To get started with the command line, you will need to learn a few basic commands.

To learn, and practice, those commands, you will need access to a running Linux distribution, or 'distro' for short. There are a number of Linux 'distros'. The distro I prefer working with is CentOS. **CentOS** is designed as an open source (free) version of **RHEL** (Red Hat Enterprise Linux), which is considered the world's leading enterprise Linux platform. The two (RHEL & CentOS) are considered functionally compatible. Being able to manage a CentOS server will be proof positive that you can manage an RHEL server. That is the reason I use CentOS.

If you do not already have access to a Linux system, I have a tutorial where I demonstrate the installation of CentOS 7 in a VirtualBox virtual machine, accessible **here**. Complete that tutorial and return here with your running CentOS virtual machine to begin this tutorial.

In this tutorial, I will go through some of the most common Linux commands that are used on a daily basis by all Linux users.

- Getting Command Help
- Navigating the File System
    - Absolute vs. Relative Paths
    - Tab Completion
- Working with Files and Directories
    - create directories
    - create files
    - copying files and directories
    - moving and renaming files and directories
    - deleting files and directories

Below is a listing of the commands we will cover in this tutorial, along with brief descriptions.

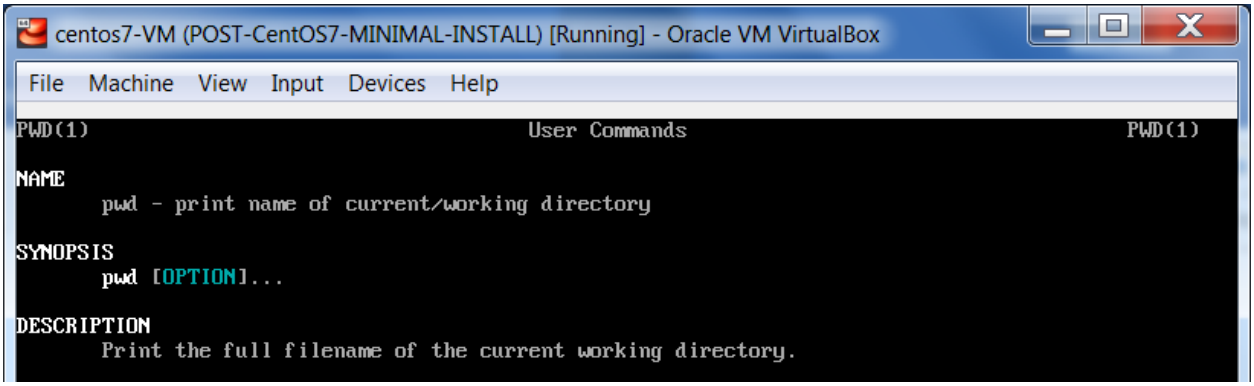
Command	Description
`man`	• display manual pages for command
`--help`	• some commands have this option that displays command usage and descriptions of the available command options
`pwd`	• present working directory
`ls`	• list the contents of directories • used to verify location of file/s
`cd`	• change working directory
`clear`	• clear the terminal screen
`touch`	• create one, or more, empty files • update access and modification times of existing file, or files.
`mkdir`	• create a directory
`rmdir`	• delete an empty directory
`cp`	• copy files and directories
`mv`	• rename or move files and directories
`rm`	• delete files or directories

Now that you have a running Linux system, are logged in and have access to the command line, we can begin.

Getting Command Help

Before we begin issuing commands, it is good to know that Linux has a few ways to get help for commands. You can use '`man <command_name>`' to get a full listing of helpful information for a specific command. For example, if we wanted to get information on the 'pwd' command, we would execute:

```
$ man pwd
```

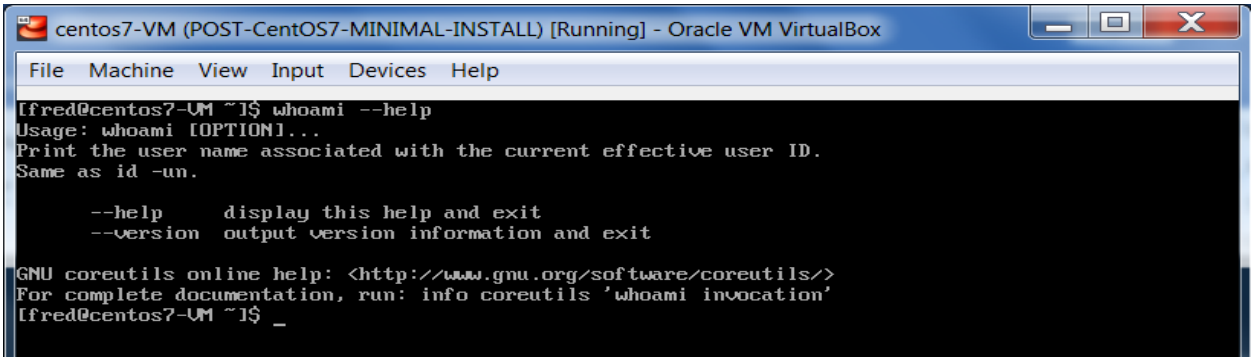


Press '**q**' to exit the man page.

Or, many commands have a '`--help`' option which usually provides you with the command usage format, as well as, descriptions of command options. To see if a command has the option, execute '`<command_name> --help`'.

For example, to see help information on the '**`whoami`**' command, execute:

```
$ whoami --help
```



Navigating the File System

Linux has a filesystem hierarchy, with the root of the filesystem being the top most directory, identified by a forward slash '**/**'. Each directory under the '**/**' root directory is separated by another forward slash.

For example: **`/etc/systemd/system`**

The root '**/**' of the filesystem contains the '**etc**' directory, and all of its contents. The '**etc**' directory contains the '**systemd**' directory, and all of its contents. And, the '**systemd**' directory contains the '**system**' directory, and all of its contents.

It will take some time to become comfortable with these concepts, but once you do, navigating the filesystem will become second nature to you.

To see where you are in the filesystem, you execute the '**pwd**' command:

```
$ pwd
```

The '**pwd**' command displays your current working directory.

In my case, since I just logged in as the regular user (also known as a **non-root** user) '**fred**', I am in the following directory:

`/home/fred`

This means I am in the home directory for user '**fred**'. Please note that the **root** user, with a home directory of **`/root`**, has all the privileges required to do anything, and everything, on the system. But, privileges are for a later tutorial.

As I mentioned in the previous section, Getting Command Help, many commands have a `--help` option. This will provide us with general command usage information, along with descriptions of command options.

To display help for the **`ls`** command, enter the following:

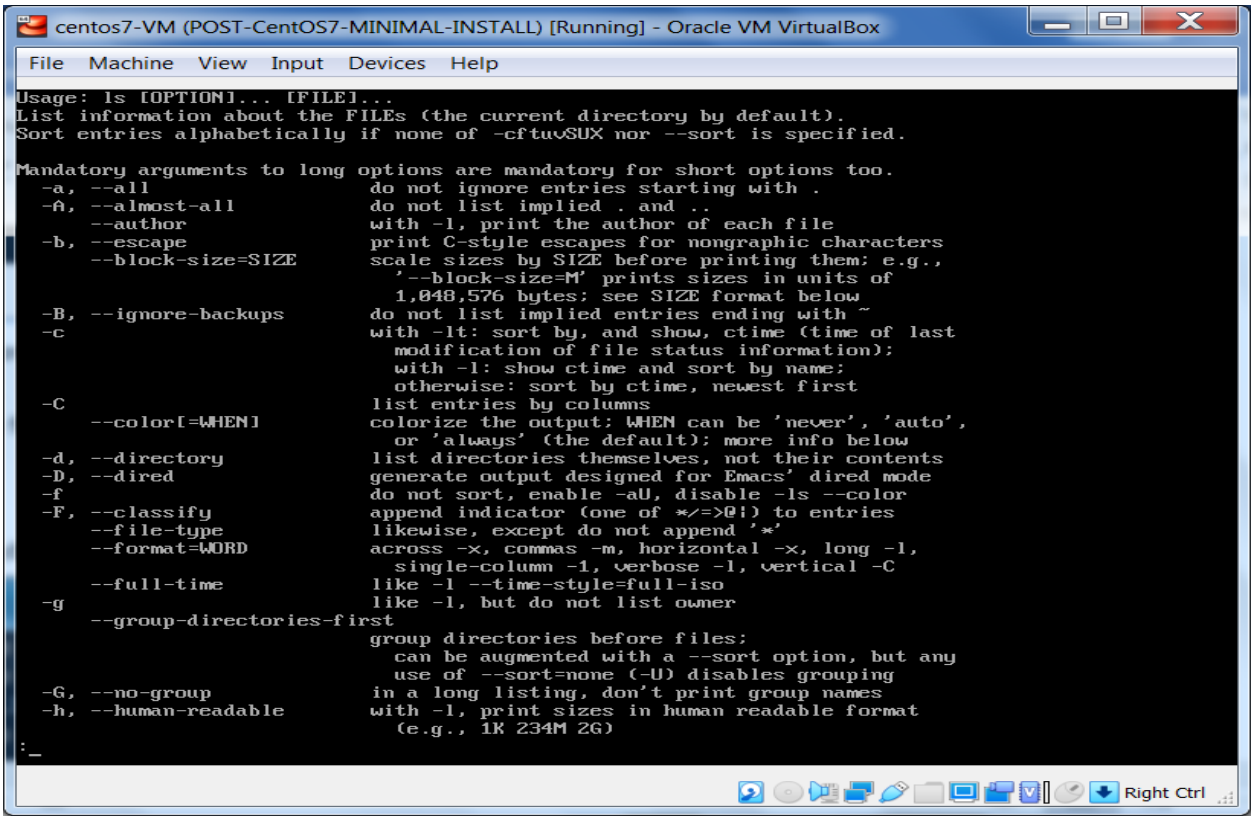
```
$ ls --help
```

You will notice a lot of information was returned in the results, and, unfortunately, there is no way of scrolling back to the beginning of the output.

We will fix that by executing the following:

```
$ ls --help | less
```

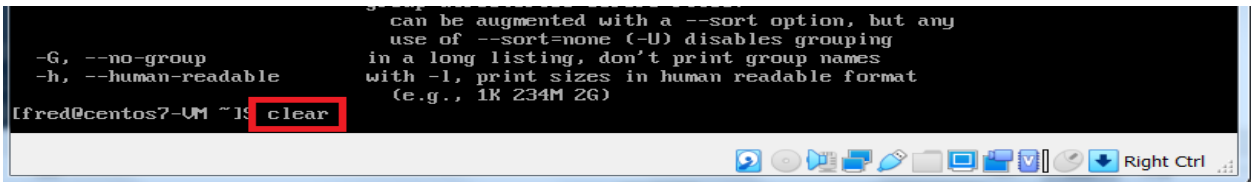
You will notice that I added a pipe '**|**', which is used to redirect the results of '**`ls --help`**' to the **`less`** command, which allows for navigation of the results using a keyboard's up/down keys.

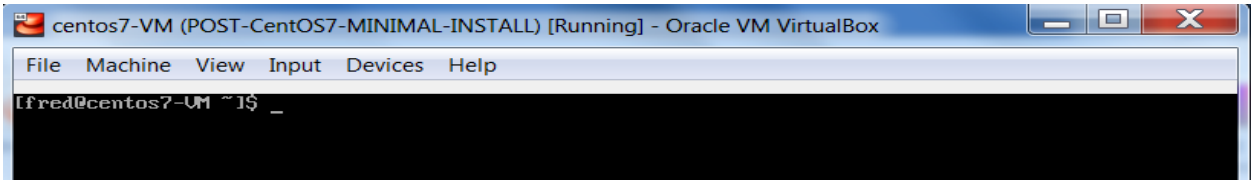


Notice that you can use your keyboard's spacebar, and up/down keys, to navigate the results. Press '**q**' to exit the help page.

Next, we will execute the `clear` command to clear our terminal:

```
$ clear
```

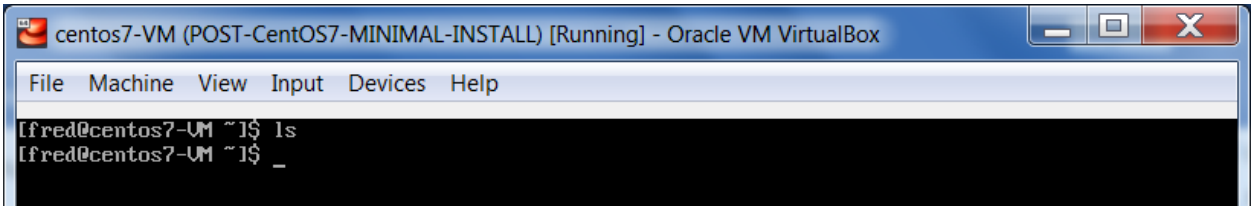




We will now list the contents of our present working directory, by executing the following:

```
$ ls
```

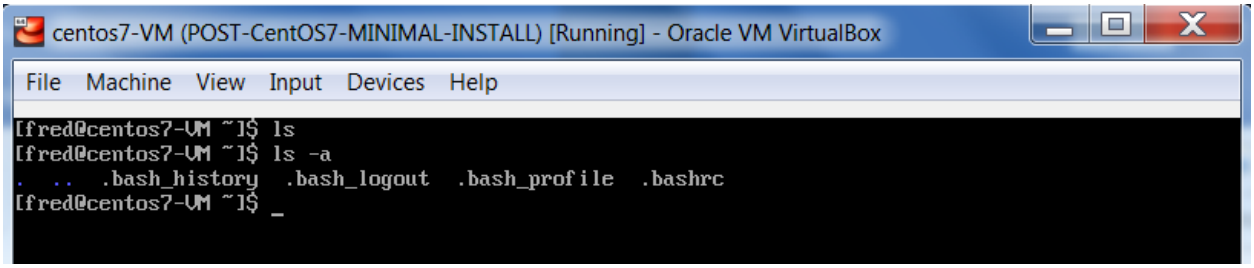
If you have just finished my other tutorial, **CentOS 7 Server Install**, executing the '**ls**' command, with no options, will return no results.



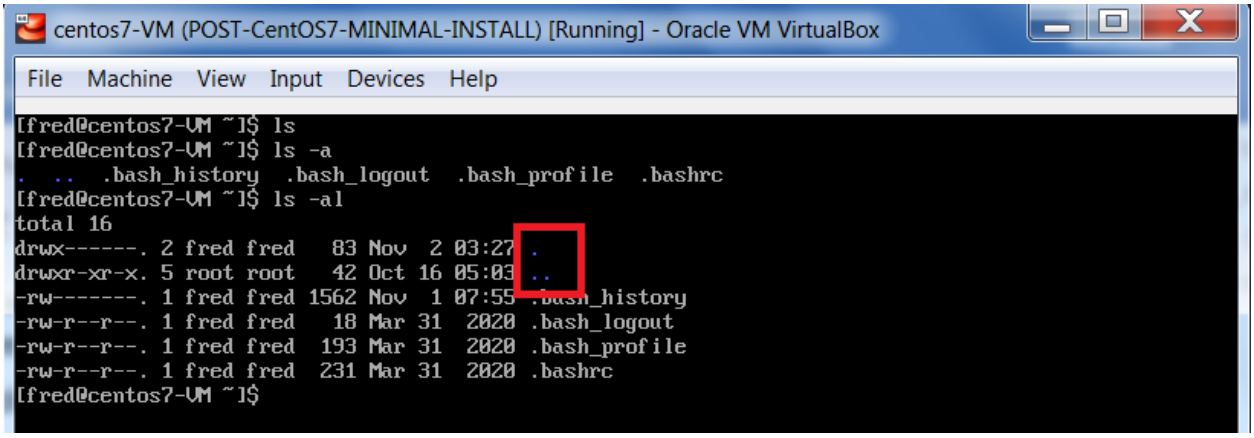
The '**ls**' command lists all available files, and directories, that are not hidden. In Linux, hidden files can be used to store user preferences and are identified by a leading dot, such as **.bash_profile**

If we want the hidden files included in the directory listing, we would execute:

```
$ ls -a
```



Now the hidden files are included, but we can improve the results by using the **-l** option. Using the `ls` command's `-l` option will return more information in a long listing format.



You will also notice, included in the results, two entries: '**.**' & '**..**'

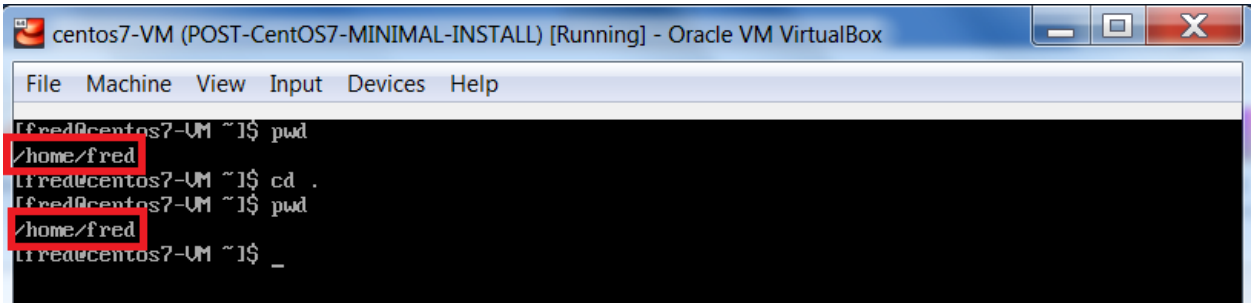
The single dot '**.**' refers to the current directory, while the double dots '**..**' refer to the parent directory.

In my case, my home directory, **/home/fred** is the current directory '.', and **/home** is the parent directory '..'.

Let's verify that. Here we will be using the '**cd**' command, which is used to change our working directory.

Execute the following commands, preceded by a dollar sign (**$**) (one after the other):

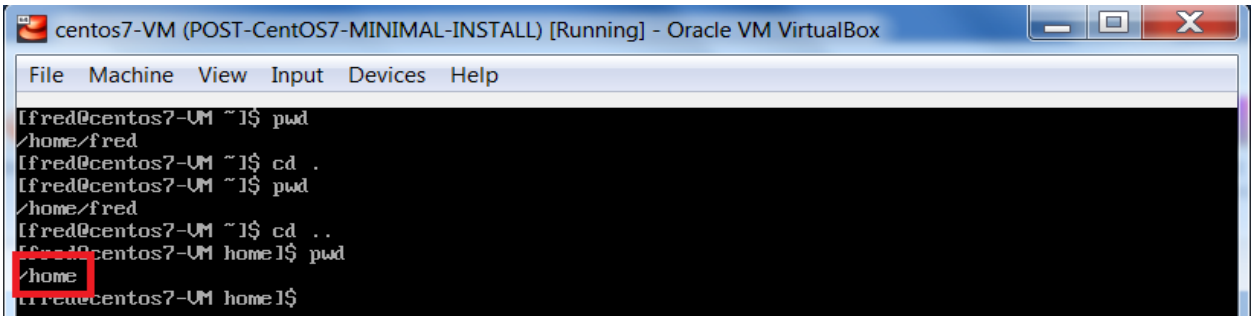
```
$ pwd          // display current working directory
/home/fred
$ cd .         // change to current directory (NOTE: no change occurs)
$ pwd          // display current working directory
/home/fred
```



Notice that our current working directory did not change.

Let's continue by executing the following:

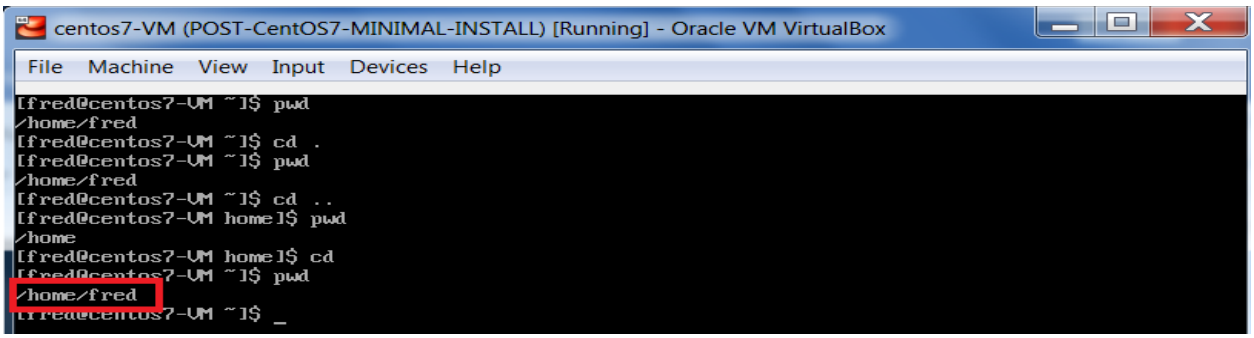
```
$ cd ..        // change to parent directory
$ pwd          // display current working directory
/home
```



Here, notice that we changed to the parent directory (**/home**).

Let's finish our verification by executing the following:

```
$ cd           // if no command argument, by default, change to user's home directory
$ pwd          // display current working directory
/home/fred
```



We confirmed that executing the '**cd**' command *sans* argument, will change to the user's home directory.

Absolute vs. Relative Paths

Before continuing, we should discuss the difference between **absolute** and **relative** paths.

An example of an absolute path would be: **/usr/bin** OR **/usr/sbin**

An example of a relative path would be: **../sbin**

All of the commands covered in this tutorial, reside in the /usr/bin directory. This directory contains executables (ls, cd, pwd, mkdir, etc...) accessible by all users on the system. Whereas, the /usr/sbin directory contains the executables used to manage the system and usually require elevated privileges to execute.

Using an absolute path, I would change directory ('**cd**') to **/usr/bin**, by executing:

```
$ cd /usr/bin          // change directory using absolute path
$ pwd
/usr/bin
```

Using a relative path, I would change directory ('**cd**') to **/usr/sbin**, by executing:

```
$ cd ../sbin           // change directory using relative path
$ pwd
/usr/sbin
```

Changing to a different directory using an absolute path forces you to type out the entire path to the destination directory, in this case, **/usr/bin**. However, using a relative path, **../sbin**, you would first specify the double dot '**..**' to move back to the parent directory /usr, before entering a forward slash '**/**', followed by an available destination directory, in this case **sbin**. The command will succeed because both directories, **bin & sbin,** have the same parent directory, **/usr**.

```
[fred@centos7-VM ~]$ ls --help | grep -e "-d,"
  -d, --directory            list directories themselves, not their contents
[fred@centos7-VM ~]$ ls -ld /usr/bin /usr/sbin
dr-xr-xr-x. 2 root root 20480 Nov  1 06:36 /usr/bin
dr-xr-xr-x. 2 root root 12288 Oct 16 05:30 /usr/sbin
```

Let's practice to become more comfortable with navigating the filesystem using both absolute and relative paths.

We will start by identifying two other absolute paths on the system

```
/etc/yum.repos.d/                    // location of configuration files for yum repositories
                                     // yum command used to install packages (programs)
/etc/systemd/system/multi-user.target.wants   // systemd service listing for multi-user target
                                              // multi-user == text mode (no GUI)
```

Now you can follow along by executing each command (preceded by a '**$**') starting on the left-hand side.

```
$ cd /etc/systemd/system/multi-user.target.wants
$ pwd
/etc/systemd/system/multi-user.target.wants
$ cd ..
$ pwd
/etc/systemd/system
$ cd ../..
$ pwd
/etc
$ cd yum.repos.d
/etc/yum.repos.d
$ cd ../systemd/system
$ pwd
/etc/systemd/system
$ cd /etc/yum.repos.d
$ pwd
/etc/yum.repos.d
```

```
$ cd ../../usr/sbin
$ pwd
/usr/sbin
$ cd ../bin
$ pwd
/usr/bin
$ cd /etc/systemd
$ pwd
/etc/systemd
$ cd system
$ pwd
/etc/systemd/system
$ cd ../../yum.repos.d
$ pwd
/etc/yum.repos.d
$ cd
$ pwd
/home/fred
```

Knowing how to quickly navigate the filesystem will become easier in time, as long as you practice regularly.

Tab Completion

Before we continue onto the next section, I would like to share with you a couple tricks I use to speed things up on the command line: the keyboard's up/down keys and tab completion. Use your keyboard's up/down keys to retrieve the most recently used commands (**history** command will be covered in my next tutorial). Use tab completion when you want to access a resource (be that a command, filename or directory) and want to reduce the number of keystrokes performed to access it.

For example, let's say you want to see how one of the system's network interfaces is configured. First, you would have to change to the following directory, which contains the network interface configuration scripts, one for each interface (also contains commands to manage the interfaces):

```
/etc/sysconfig/network-scripts/              // contains network config commands & scripts
```

Using tab completion, at the command line, you would start by typing:

```
$ cd /etc/sysc<TAB><TAB>
sysconfig/   sysctl.conf  sysctl.d/
```

You will notice that there are three files in the 'etc' directory that begin with '**sysc**', so hitting the <TAB> key once, will return nothing, but hitting the <TAB> key a second time, will display, in this case, the three files that begin with '**sysc**'. Next step will be to enter the minimum text required to make the search unique. In this case, we only have to enter a single character, 'o', and hit the <TAB> key once:

```
$ cd /etc/sysco<TAB>
```

which will complete the directory name I want, and allow me to continue:

```
$ cd /etc/sysconfig/network<TAB><TAB>
```

Again, once I reached '/network', I hit <TAB> once, with no returns, so I hit <TAB> a second time, which resulted in:

```
$ cd /etc/sysconfig/network<TAB><TAB>
network          network-scripts/
```

I see that all I need to add is the '-' (dash) to make the search unique, followed by the <TAB> key:

```
$ cd /etc/sysconfig/network-<TAB>
```

which results in the destination directory I want

```
$ cd /etc/sysconfig/network-scripts/
$ pwd
/etc/sysconfig/network-scripts
```

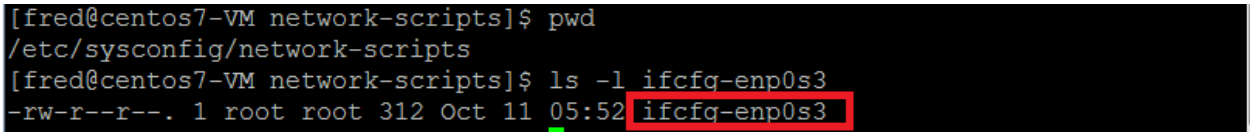
Once in the directory, I can perform a directory listing ('**ls**') to locate the file I want. In this case, I know the file starts with '**ifcfg-**', but I cannot remember the entire filename. So, I will use tab completion again.

Again, I enter '**ifcfg-**', hit <TAB> once, with no returns, so I hit <TAB> a second time, which resulted in:

```
$ ls -l ifcfg-<TAB><TAB>
ifcfg-enp0s3  ifcfg-lo
```

From the results, I see a filename which includes the interface name (**enp0s3**). To make my search unique, I have to enter 'e', followed by the <TAB> key:

```
$ ls -l ifcfg-e<TAB>
$ ls -l ifcfg-enp0s3
-rw-r--r--. 1 root root 312 Oct 11 05:52 ifcfg-enp0s3
```



It will take you a while to learn where everything is on a Linux system, but tab completion is a great way to reduce the number of keystrokes required to access the resources you need.

Working with Files and Directories

In this section, you will learn how to work with files and directories. More specifically, I will demonstrate the commands used to:

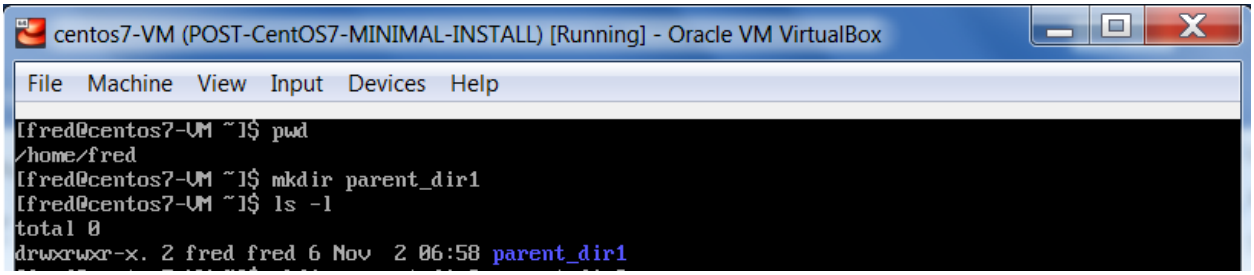
- create directories
- create files
- copy files and directories
- move and rename files and directories
- delete files and directories

Create Directories

We will start by creating new directories using the '**mkdir**' command.

Ensure you are in your home directory (remember '**cd**' will get you there). Then, from the command line, execute the following (one command after the other):

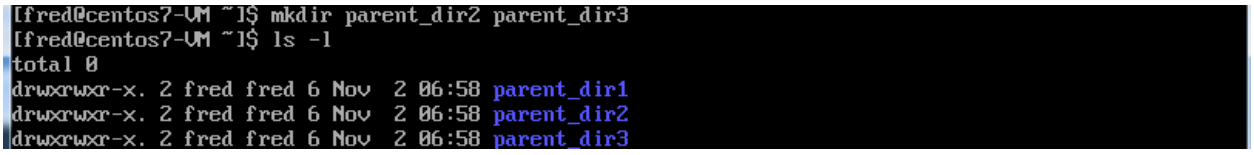
```
$ mkdir parent_dir1
$ ls -l
```



We have successfully created our first directory.

Now, let's create multiple directories at once by executing:

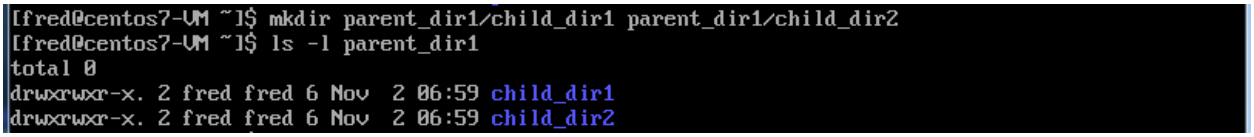
```
$ mkdir parent_dir2 parent_dir3
$ ls -l
```



You will notice that we can provide multiple arguments to the '**mkdir**' command at once, to create different directories, instead of having to execute two separate '**mkdir**' commands.

We can now create child directories for one of our newly created directories by executing the following:

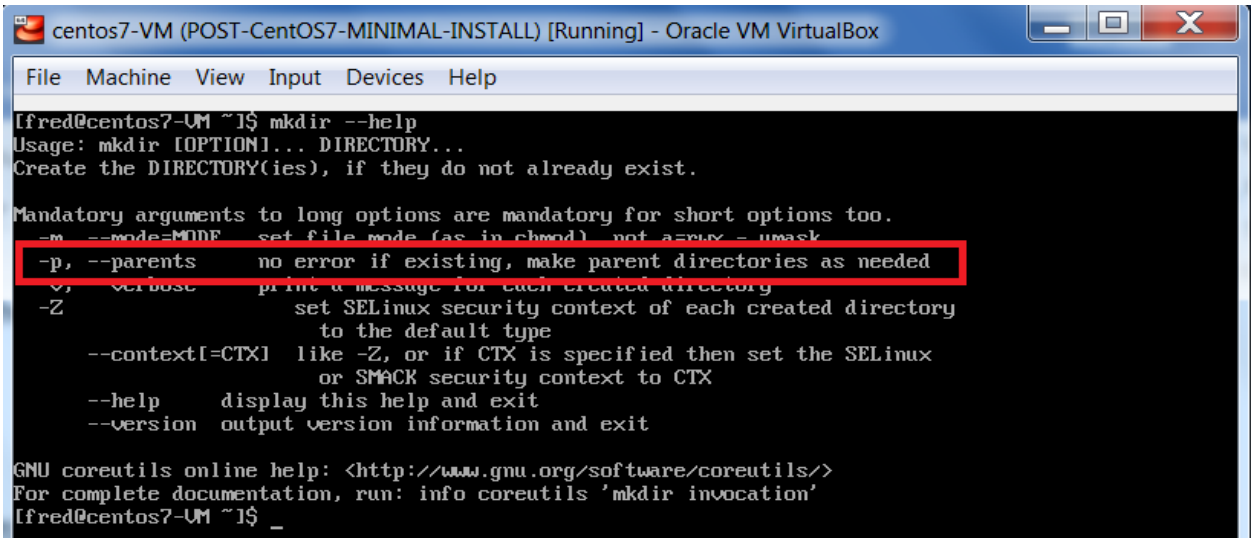
```
$ mkdir parent_dir1/child_dir1 parent_dir1/child_dir2
$ ls -l parent_dir1
```



The directory, **parent_dir1**, is considered the parent of both **child_dir1 & child_dir2**.

What If we need to create a multi-level directory structure, such as `parent_dir4/child_dir1/mini_dir1`, without having to execute three separate '**mkdir**' commands. Is that possible? Let's first check to see if any help is available to us by executing:

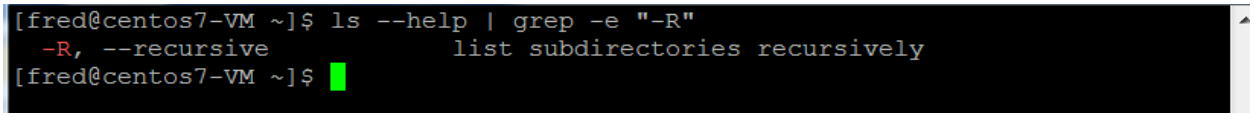
```
$ mkdir --help
```



Looks like we can use the **-p** option of the '**mkdir**' command. Execute the following command:

```
$ mkdir -p parent_dir4/child_dir1/mini_dir1
```

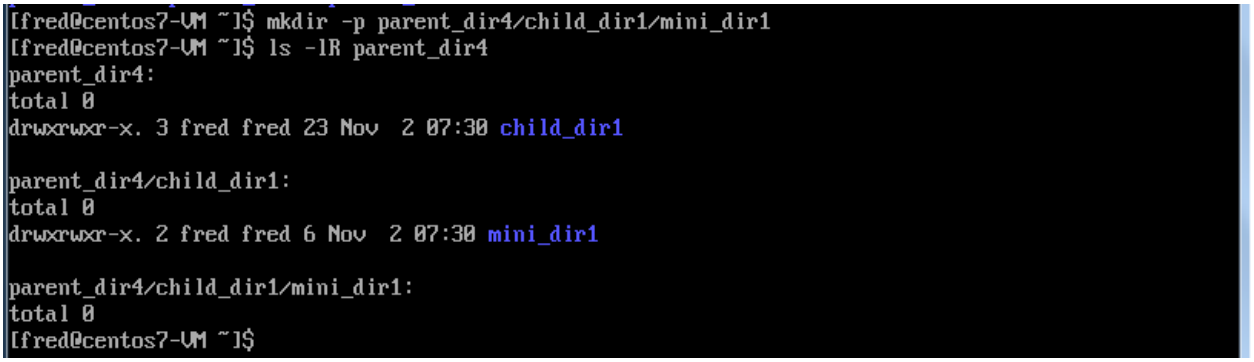
To confirm that we've created the new directory structure, we will use the **-R** option, of the '**ls**' command, which lists subdirectories recursively. Again, I used the **--help** option of the '**ls**' command to see if that option was available.



For now, please disregard my use of the '**grep**' command (**grep** is used to search for patterns). It will be covered in a future tutorial.

Again, to confirm that we've created the new directory structure, execute the following:

```
$ ls -lR parent_dir4
```



We see that the multi-level directory structure has been successfully created using only one command.

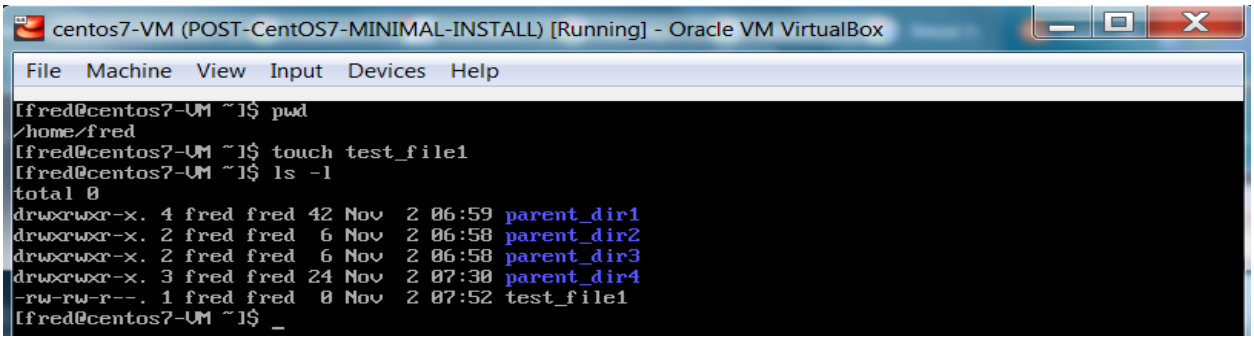
Now that you have a basic understanding of directories, and how to create them, we will move onto the file creation section.

Create Files

To begin, we will be creating a number of empty files using the '**touch**' command.

Ensure you are in your home directory (remember '**cd**' will get you there). Then, from the command line, execute the following (one after the other):

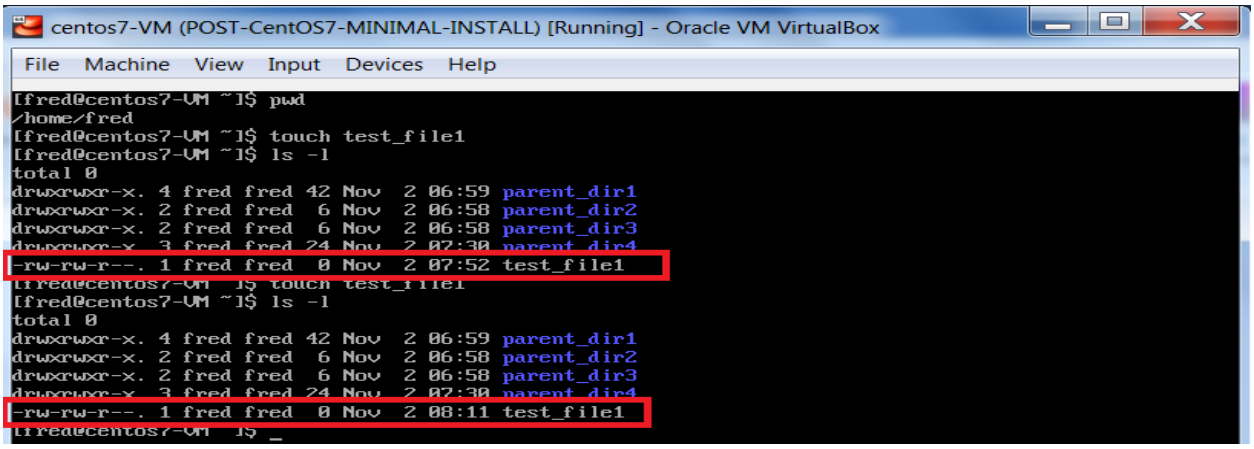
```
$ touch test_file1
$ ls -l
```



You will notice that we successfully created a file. The '**touch**' command is also used to change file access and modification timestamps. If a user creates an empty file, using '**touch**', the file's access, and modification, timestamps can be changed when the user has content to add to the file.

Let's confirm this by issuing the following command:

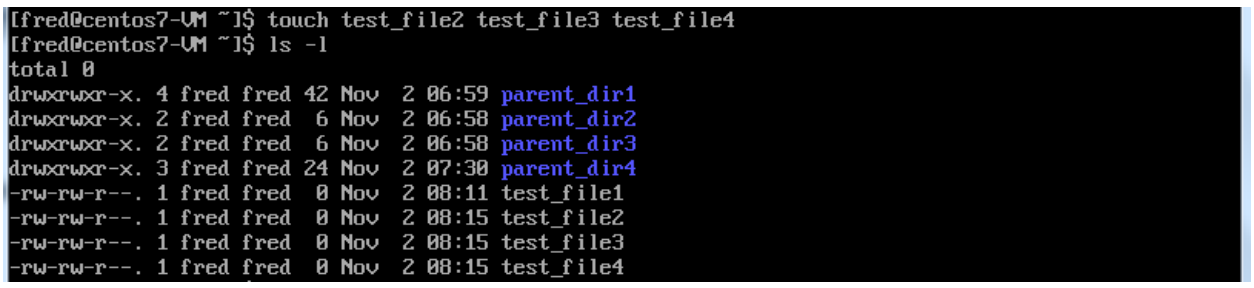
```
$ touch test_file1
$ ls -l
```



You will notice that **test_file1's** timestamp has been updated.

We can also create multiple files at once using '**touch**'. Execute the following:

```
$ touch test_file2 test_file3 test_file4
$ ls -l
```



Now, let's populate our directories with some files by executing the following:

```
$ touch parent_dir1/child_dir1/test_file1
$ touch parent_dir1/child_dir1/test_file2
$ ls -lR parent_dir1
```

```
[fred@centos7-VM ~]$ touch parent_dir1/child_dir1/test_file1
[fred@centos7-VM ~]$ touch parent_dir1/child_dir1/test_file2
[fred@centos7-VM ~]$ ls -lR parent_dir1
parent_dir1:
total 0
drwxrwxr-x. 2 fred fred 42 Nov  2 08:19 child_dir1
drwxrwxr-x. 2 fred fred  6 Nov  2 06:59 child_dir2

parent_dir1/child_dir1:
total 0
-rw-rw-r--. 1 fred fred 0 Nov  2 08:19 test_file1
-rw-rw-r--. 1 fred fred 0 Nov  2 08:19 test_file2

parent_dir1/child_dir2:
total 0
[fred@centos7-VM ~]$ _
```

Now that we have our directories populated with files, we can move to the next section.

Copying Files and Directories

We will begin this section by listing the contents of our home directory, recursively, to see what we have to work with. Ensure you are in your home directory and execute the following:

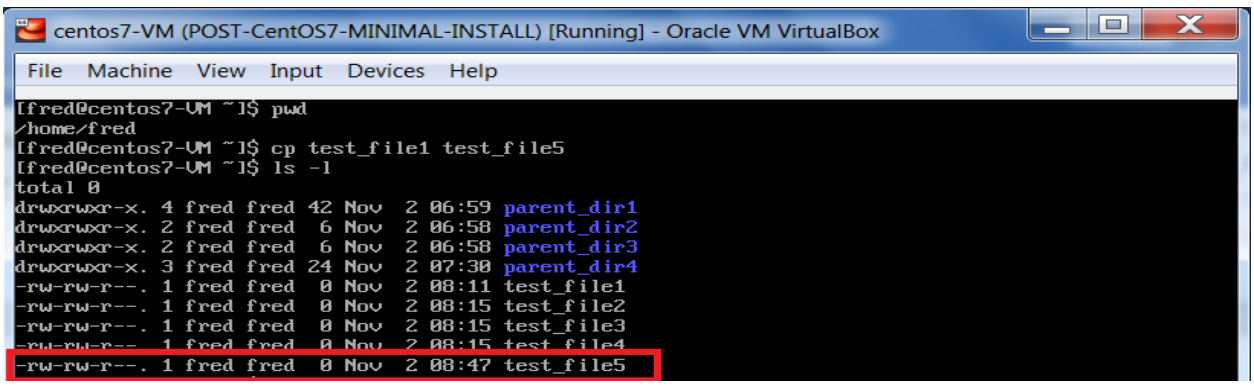
```
$ ls -R | less
```

```
[fred@centos7-VM ~]$ pwd
/home/fred
[fred@centos7-VM ~]$ ls -R
.:
parent_dir1  parent_dir2  parent_dir3  parent_dir4  test_file1  test_file2  test_file3  test_file4

./parent_dir1:
child_dir1  child_dir2

./parent_dir1/child_dir1:
test_file1  test_file2

./parent_dir1/child_dir2:

./parent_dir2:

./parent_dir3:

./parent_dir4:
child_dir1

./parent_dir4/child_dir1:
mini_dir1

./parent_dir4/child_dir1/mini_dir1:
[fred@centos7-VM ~]$
```

Please note that I didn't use the **-l** option, for the '**ls**' command, this time, to reduce the image size.

We have a few directories to work with. First, we will make a copy of a file, to the same directory, by renaming it. Execute the following:

```
$ cp test_file1 test_file5
$ ls -l
```



The copy operation was a success. Next, we will copy a file from our home directory to one of our empty directories. Execute the following:

```
$ cp test_file2 parent_dir2
```

```
$ ls -l parent_dir2
```

```
[fred@centos7-VM ~]$ cp test_file2 parent_dir2
[fred@centos7-VM ~]$ ls -l parent_dir2
total 0
-rw-rw-r--. 1 fred fred 0 Nov  2 08:49 test_file2
[fred@centos7-VM ~]$ _
```

Now, let's copy the contents of one directory into another. Note that I will be using the asterisk '*****' wildcard character which, in this case, represents the entire directory contents. Execute the following:

```
$ cp parent_dir1/child_dir1/* parent_dir1/child_dir2
```

```
$ ls -lR parent_dir1
```

```
[fred@centos7-VM ~]$ cp parent_dir1/child_dir1/* parent_dir1/child_dir2/
[fred@centos7-VM ~]$ ls -lR parent_dir1
parent_dir1:
total 0
drwxrwxr-x. 2 fred fred 42 Nov  2 08:19 child_dir1
drwxrwxr-x. 2 fred fred 42 Nov  2 08:59 child_dir2

parent_dir1/child_dir1:
total 0
-rw-rw-r--. 1 fred fred 0 Nov  2 08:19 test_file1
-rw-rw-r--. 1 fred fred 0 Nov  2 08:19 test_file2

parent_dir1/child_dir2:
total 0
-rw-rw-r--. 1 fred fred 0 Nov  2 08:59 test_file1
-rw-rw-r--. 1 fred fred 0 Nov  2 08:59 test_file2
```

Now, both directories contain **test_file1 & test_file2.**

To finish off this section, we will copy a directory, including its contents, into another directory. To do this we must use the -R option of the '**cp**' command.

```
[fred@centos7-VM ~]$ cp --help | grep -e "-R"
  -R, -r, --recursive          copy directories recursively
[fred@centos7-VM ~]$
```

Before performing the copy operation, let's list the contents of both directories, for verification purposes afterwards, by executing the following:

```
$ ls -l parent_dir2
```

```
$ ls -lR parent_dir1
```

```
[fred@centos7-VM ~]$ ls -l parent_dir2
total 0
-rw-rw-r--. 1 fred fred 0 Nov  2 08:49 test_file2
[fred@centos7-VM ~]$ ls -lR parent_dir1
parent_dir1:
total 0
drwxrwxr-x. 2 fred fred 42 Nov  2 08:19 child_dir1
drwxrwxr-x. 2 fred fred 42 Nov  2 08:59 child_dir2

parent_dir1/child_dir1:
total 0
-rw-rw-r--. 1 fred fred 0 Nov  2 08:19 test_file1
-rw-rw-r--. 1 fred fred 0 Nov  2 08:19 test_file2

parent_dir1/child_dir2:
total 0
-rw-rw-r--. 1 fred fred 0 Nov  2 08:59 test_file1
-rw-rw-r--. 1 fred fred 0 Nov  2 08:59 test_file2
[fred@centos7-VM ~]$
```

Now that we have a reference point, we can execute the following:

```
$ cp -R parent_dir2 parent_dir1/
$ ls -lR parent_dir1
```

```
[fred@centos7-VM ~]$ cp -R parent_dir2 parent_dir1/
[fred@centos7-VM ~]$ ls -lR parent_dir1
parent_dir1:
total 0
drwxrwxr-x. 2 fred fred 42 Nov  2 08:19 child_dir1
drwxrwxr-x. 2 fred fred 42 Nov  2 08:59 child_dir2
drwxrwxr-x. 2 fred fred 24 Nov  2 09:04 parent_dir2

parent_dir1/child_dir1:
total 0
-rw-rw-r--. 1 fred fred 0 Nov  2 08:19 test_file1
-rw-rw-r--. 1 fred fred 0 Nov  2 08:19 test_file2

parent_dir1/child_dir2:
total 0
-rw-rw-r--. 1 fred fred 0 Nov  2 08:59 test_file1
-rw-rw-r--. 1 fred fred 0 Nov  2 08:59 test_file2

parent_dir1/parent_dir2:
total 0
-rw-rw-r--. 1 fred fred 0 Nov  2 09:04 test_file2
[fred@centos7-VM ~]$ _
```

We have successfully copied a directory, including its contents, into another directory. Now that you are becoming more comfortable working with files and directories, you should be ready for the next section.

Moving and Renaming Files and Directories

To begin this section, ensure that you are in your home directory ('**cd**' if needed), and execute a directory listing:

```
$ ls -l
```

```
[fred@centos7-VM ~]$ pwd
/home/fred
[fred@centos7-VM ~]$ ls -l
total 0
drwxrwxr-x. 4 fred fred 42 Nov  2 09:07 parent_dir1
drwxrwxr-x. 2 fred fred 24 Nov  2 08:49 parent_dir2
drwxrwxr-x. 2 fred fred  6 Nov  2 06:58 parent_dir3
drwxrwxr-x. 3 fred fred 24 Nov  2 07:30 parent_dir4
-rw-rw-r--. 1 fred fred  0 Nov  2 08:11 test_file1
-rw-rw-r--. 1 fred fred  0 Nov  2 08:15 test_file2
-rw-rw-r--. 1 fred fred  0 Nov  2 08:15 test_file3
-rw-rw-r--. 1 fred fred  0 Nov  2 08:15 test_file4
-rw-rw-r--. 1 fred fred  0 Nov  2 08:47 test_file5
```

Our first operation will be to rename a file using the '**mv**' command by executing the following:

```
$ mv test_file1 test_file6
$ ls -l test_file*
```

```
[fred@centos7-VM ~]$ mv test_file1 test_file6
[fred@centos7-VM ~]$ ls -l test_file*
-rw-rw-r--. 1 fred fred 0 Nov  2 08:15 test_file2
-rw-rw-r--. 1 fred fred 0 Nov  2 08:15 test_file3
-rw-rw-r--. 1 fred fred 0 Nov  2 08:15 test_file4
-rw-rw-r--. 1 fred fred 0 Nov  2 08:47 test_file5
-rw-rw-r--. 1 fred fred 0 Nov  2 08:11 test_file6
```

We successfully renamed a file. Note my use of the asterisk '*****' wildcard character in my directory listing. In this instance, the '*' wildcard represents zero or more characters. Any file that begins with '**test_file**' will be returned in the results. Just wanted to shorten my results for a smaller image.

To determine the '**mv**' command options available , I executed:

```
$ mv --help | less
```

Please note that my screenshot, once again, includes the '**grep**' command, which will be covered in another tutorial. Also note that the '**mv**' command has more options, but these are the ones I wanted to focus on.

```
[jack@centos7-VM ~]$ mv --help | grep -e "-[f,i,n],"
  -f, --force                   do not prompt before overwriting
  -i, --interactive             prompt before overwrite
  -n, --no-clobber              do not overwrite an existing file
If you specify more than one of -i, -f, -n, only the final one takes effect.
```

Again, let's see what we have to work with by executing the following:

```
$ ls -l parent_dir1/child_dir2
```

```
$ ls -l parent_dir2
```

```
[fred@centos7-VM ~]$ ls -l parent_dir1/child_dir2
total 0
-rw-rw-r--. 1 fred fred 0 Nov  2 08:59 test_file1
-rw-rw-r--. 1 fred fred 0 Nov  2 08:59 test_file2
[fred@centos7-VM ~]$ ls -l parent_dir2
total 0
-rw-rw-r--. 1 fred fred 0 Nov  2 08:19 test_file2
```

Note the different timestamps for '**test_file2**'.

Now, to move the contents of one directory to another using the **-i (interactive)** option (which will prompt us to confirm the operation before execution), execute the following:

```
$ mv -i parent_dir1/child_dir2/* parent_dir2
```

When prompted to overwrite '**test_file2**', enter '**n**' (**NO**).

```
[fred@centos7-VM ~]$ mv -i parent_dir1/child_dir2/* parent_dir2
mv: overwrite 'parent_dir2/test_file2'? n
[fred@centos7-VM ~]$
```

The operation completed successfully. Now, we will perform our verification.

```
$ ls -l parent_dir1/child_dir2
```

```
$ ls -l parent_dir2
```

```
[fred@centos7-VM ~]$ ls -l parent_dir1/child_dir2
total 0
-rw-rw-r--. 1 fred fred 0 Nov  2 08:59 test_file2
[fred@centos7-VM ~]$ ls -l parent_dir2
total 0
-rw-rw-r--. 1 fred fred 0 Nov  2 08:59 test_file1
-rw-rw-r--. 1 fred fred 0 Nov  2 08:19 test_file2
[fred@centos7-VM ~]$
```

Note that only one file was moved, '**test_file1**', and that '**test_file2**' was **NOT** overwritten.

We will now use the **-n (no clobber)** option to ensure files are not overwritten, but without prompting the user.

Again, use the image above as a reference when performing verification, and execute the following:

```
$ mv -n parent_dir1/child_dir2/* parent_dir2
$ ls -l parent_dir1/child_dir2
$ ls -l parent_dir2
```

```
[fred@centos7-VM ~]$ mv -n parent_dir1/child_dir2/* parent_dir2
[fred@centos7-VM ~]$ ls -l parent_dir1/child_dir2
total 0
-rw-rw-r--. 1 fred fred 0 Nov  2 08:59 test_file2
[fred@centos7-VM ~]$ ls -l parent_dir2
total 0
-rw-rw-r--. 1 fred fred 0 Nov  2 08:59 test_file1
-rw-rw-r--. 1 fred fred 0 Nov  2 08:19 test_file2
```

Note that no file was moved. Timestamp for '**test_file2**' did **NOT** change.

We will now use the **-f (force)** option which will overwrite a file if it already exists.

Again, use the image above as a reference when performing verification, and execute the following:

```
$ mv -f parent_dir1/child_dir2/test_file2 parent_dir2
$ ls -l parent_dir1/child_dir2
$ ls -l parent_dir2
```

```
[fred@centos7-VM ~]$ mv -f parent_dir1/child_dir2/test_file2 parent_dir2
[fred@centos7-VM ~]$ ls -l parent_dir2
total 0
-rw-rw-r--. 1 fred fred 0 Nov  2 08:59 test_file1
-rw-rw-r--. 1 fred fred 0 Nov  2 08:59 test_file2
```

Note that the file was moved. Timestamp for '**test_file2**' did change.

Our final '**mv**' operation will be to move two directories to a different destination directory. Please note that the destination already has a directory with the same name as one of the source directories, and will **NOT** be overwritten. Therefore, only one directory will be moved. We will first get a reference to perform our post-operation verification, by executing:

```
$ ls -lR parent_dir1
$ ls -lR parent_dir4
```

```
[fred@centos7-VM ~]$ ls -lR parent_dir1
parent_dir1:
total 0
drwxrwxr-x. 2 fred fred 42 Nov  3 03:45 child_dir1
drwxrwxr-x. 2 fred fred 24 Nov  3 03:35 child_dir2

parent_dir1/child_dir1:
total 0
-rw-rw-r--. 1 fred fred 0 Nov  3 03:45 test_file1
-rw-rw-r--. 1 fred fred 0 Nov  3 03:45 test_file2

parent_dir1/child_dir2:
total 0
-rw-rw-r--. 1 fred fred 0 Nov  2 08:59 test_file2
```

```
[fred@centos7-VM ~]$ ls -lR parent_dir4
parent_dir4:
total 0
drwxrwxr-x. 3 fred fred 23 Nov  2 07:30 child_dir1

parent_dir4/child_dir1:
total 0
drwxrwxr-x. 2 fred fred 6 Nov  2 07:30 mini_dir1

parent_dir4/child_dir1/mini_dir1:
total 0
```

You will notice that both directories contain a '**child_dir1**' directory. To perform the directory move operation, ensure you are in your home directory ('**cd**' if need be) and execute the following:

```
$ cd parent_dir1                              // to move into the 'parent_dir1' directory
$ mv child_dir1 child_dir2 ../parent_dir4  // move directories (note use of relative path)
```

```
[fred@centos7-VM ~]$ pwd
/home/fred
[fred@centos7-VM ~]$ cd parent_dir1
[fred@centos7-VM parent_dir1]$ pwd
/home/fred/parent_dir1
[fred@centos7-VM parent_dir1]$ mv child_dir1 child_dir2 ../parent_dir4
mv: cannot move 'child_dir1' to '../parent_dir4/child_dir1': File exists
[fred@centos7-VM parent_dir1]$
```

Note the warning and verify our results by executing the following:

```
$ cd                                  // to return to my home directory
$ ls -lR parent_dir1                  // recursively list directory contents for verification
$ ls -lR parent_dir4
```

```
[fred@centos7-VM parent_dir1]$ cd
[fred@centos7-VM ~]$ pwd
/home/fred
[fred@centos7-VM ~]$ ls -lR parent_dir1
parent_dir1:
total 0
drwxrwxr-x. 2 fred fred 42 Nov  3 03:45 child_dir1

parent_dir1/child_dir1:
total 0
-rw-rw-r--. 1 fred fred 0 Nov  3 03:45 test_file1
-rw-rw-r--. 1 fred fred 0 Nov  3 03:45 test_file2
[fred@centos7-VM ~]$
```

```
[fred@centos7-VM ~]$ ls -lR parent_dir4
parent_dir4:
total 0
drwxrwxr-x. 3 fred fred 23 Nov  2 07:30 child_dir1
drwxrwxr-x. 2 fred fred  6 Nov  3 05:11 child_dir2

parent_dir4/child_dir1:
total 0
drwxrwxr-x. 2 fred fred 6 Nov  2 07:30 mini_dir1

parent_dir4/child_dir1/mini_dir1:
total 0

parent_dir4/child_dir2:
total 0
```

In this instance, the '**child_dir1**' of '**parent_dir4**' was **NOT** overwritten, but '**child_dir2**' was moved to the destination directory, '**parent_dir4**'.

Deleting Files and Directories

In this section we will be using the '**rm**' and '**rmdir**' commands. The '**rmdir**' command can be used to delete empty directories, while the '**rm**' command can be used to delete files, as well as, non-empty directories.

We will begin by using the 'rmdir' command to try removing a non-empty directory. First, ensure you are in your home directory ('**cd**') and execute the following:

```
$ ls -lR parent_dir1                  // recursively list directory contents for verification
$ rmdir parent_dir1/child_dir1        // attempt to delete non-empty directory
```

```
[fred@centos7-VM ~]$ ls -lR parent_dir1
parent_dir1:
total 0
drwxrwxr-x. 2 fred fred 42 Nov  3 03:45 child_dir1

parent_dir1/child_dir1:
total 0
-rw-rw-r--. 1 fred fred 0 Nov  3 03:45 test_file1
-rw-rw-r--. 1 fred fred 0 Nov  3 03:45 test_file2
[fred@centos7-VM ~]$ rmdir parent_dir1/child_dir1/
rmdir: failed to remove 'parent_dir1/child_dir1/': Directory not empty
[fred@centos7-VM ~]$
```

Just wanted to show you what happens when we try removing a non-empty directory.

Next step will be to delete the contents of the '**child_dir1**' directory using the '**rm**' command, and then execute '**rmdir**' again:

```
$ ls -l parent_dir1/child_dir1           // directory listing for verification
$ rm parent_dir1/child_dir1/test_file*   // delete files whose names start with 'test_file'
$ ls -l parent_dir1/child_dir1
$ rmdir parent_dir1/child_dir1          // delete directory
$ ls -l parent_dir1                     // directory listing for verification
```

```
[fred@centos7-VM ~]$ ls -l parent_dir1/child_dir1/
total 0
-rw-rw-r--. 1 fred fred 0 Nov  3 03:45 test_file1
-rw-rw-r--. 1 fred fred 0 Nov  3 03:45 test_file2
[fred@centos7-VM ~]$ rm parent_dir1/child_dir1/test_file*
[fred@centos7-VM ~]$ ls -l parent_dir1/child_dir1/
total 0
[fred@centos7-VM ~]$ rmdir parent_dir1/child_dir1/
[fred@centos7-VM ~]$ ls -l parent_dir1
total 0
```

Both delete (remove) operations succeeded. Below is a listing of the '**rm**' command options we will cover.

```
[jack@centos7-VM ~]$ rm --help | grep -e '-[i] '
  -i                    prompt before every removal
[jack@centos7-VM ~]$ rm --help | grep -e '-[f,r],'
  -f, --force           ignore nonexistent files and arguments, never prompt
  -r, -R, --recursive   remove directories and their contents recursively
```

We will use the '**rm**' command to delete (remove) a non-empty directory. First, we will use the **-i (interactive)** option to be prompted before deletion and second, we will use the **-f (force)** option to avoid being prompted. Both of the operations will require the **-r**, or **-R (recursive)**, option to delete the directory, and its contents (includes any child directories along with their contents).

To delete a directory and its contents, while being prompted, execute the following:

When prompted enter '**y**' to confirm each operation.

```
$ cd                  // ensure you are in home directory
$ ls -l parent_dir2   // list directory contents
$ rm -iR parent_dir2  // delete (remove) directory and its contents interactively
$ ls -l parent_dir2   // verify directory successfully deleted
```

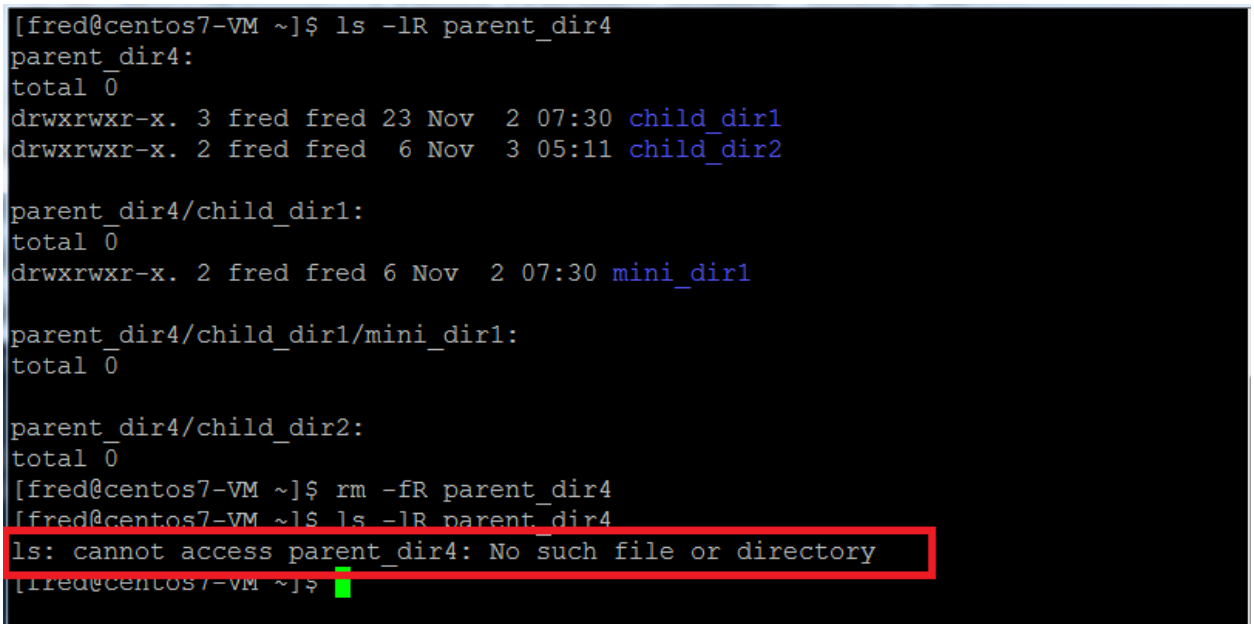
```
[fred@centos7-VM ~]$ cd
[fred@centos7-VM ~]$ ls -l parent_dir2
total 0
-rw-rw-r--. 1 fred fred 0 Nov  2 08:59 test_file1
-rw-rw-r--. 1 fred fred 0 Nov  2 08:59 test_file2
[fred@centos7-VM ~]$ rm -iR parent_dir2
rm: descend into directory 'parent_dir2'? y
rm: remove regular empty file 'parent_dir2/test_file2'? y
rm: remove regular empty file 'parent_dir2/test_file1'? y
rm: remove directory 'parent_dir2'? y
[fred@centos7-VM ~]$ ls -l parent_dir2
ls: cannot access parent_dir2: No such file or directory
[fred@centos7-VM ~]$
```

We have successfully deleted the directory and its contents.

Our final operation will be deleting a directory, its contents, including child directories and their contents.

Ensure you are in your home directory (‘cd’) and execute the following:

```
$ ls -lR parent_dir4  // list directory contents
$ rm -fR parent_dir4  // delete (remove) directory and its contents
$ ls -lR parent_dir4  // verify directory successfully deleted
```



We have successfully deleted the directory and all of its contents.

I hope you have enjoyed completing this tutorial and found it helpful.

We covered the basics of navigating the filesystem, as well as, working with files and directories. That was a good start. You might be interested in how to manage a Linux server, and its available services. Or, how to automate tasks with scripting. If you are interested in continuing your Linux learning journey, I have a number of other Linux tutorials that can be accessed here, while my main tutorials page can be accessed here.

Back to Top