Manage Files from the command line

Linux #redhat #filesystem #links #pipe

Introduction to Filesystem

- What is a Filesystem?
  - It is a system used by an operating system to manage files. The system controls how data is saved or retrieved.
  - See the filesystem as an organized closet
  - A filesystem avoids a cluttered way of organizing things
- Operating system stores files and directories in an organized and structured way
  - System configuration file = Folder A
  - User files = Folder B
  - Log files = Folder C
  - Commands or scripts = Folder D and so on
- There are many different types of filesystems. In general, improvements have been made to filesystem with new releases of operating systems and each new filesystem has been given a different name
  - Ex. ext3, ext4, xfs, NTFS, FAT etc.
    - 'ext' and 'xfs' are known to be Linux filesystems
    - 'NTFS' and 'FAT' are recognized as Windows filesystems
- With a Filesystem, the OS knows exactly which folder to look for when needed

Access your machine files

- In a Windows OS you can simply run the File Explorer built-in program and look for "This PC" in the list of directories shown at the left. Once there just click on the name of the machine and you will see all the directories branched from your machine. In Windows everything starts from the "C" drive.
- In a Linux OS you have to go into `cd /` because everything starts from the `/`, which is the root directory.

Example using `cd` command:

```
[user@localhost ~]$ cd /
[user@localhost /]$
```

- This command will change the current directory to the root directory "`/`"
- No Output

Example using `ls` command:

```
[user@localhost /]$ ls -l
```

- Use the `ls` command paired with the specifier `-l` to show a long listing format of the directories inside the current parent directory.
- `ls -l` lists different fields
- This is useful to confirm the current directory location of the terminal

Output:

```
total 24
dr-xr-xr-x.   2 root root    6 Aug  9  2021 afs
lrwxrwxrwx.   1 root root    7 Aug  9  2021 bin -> usr/bin
dr-xr-xr-x.   5 root root 4096 Jun  3 13:38 boot
drwxr-xr-x.  20 root root 3300 Jun  4 13:32 dev
drwxr-xr-x. 132 root root 8192 Jun  4 13:32 etc
drwxr-xr-x.   3 root root   20 Jun  3 13:24 home
lrwxrwxrwx.   1 root root    7 Aug  9  2021 lib -> usr/lib
lrwxrwxrwx.   1 root root    9 Aug  9  2021 lib64 -> usr/lib64
drwxr-xr-x.   2 root root    6 Aug  9  2021 media
drwxr-xr-x.   2 root root    6 Aug  9  2021 mnt
drwxr-xr-x.   2 root root    6 Aug  9  2021 opt
dr-xr-xr-x. 300 root root    0 Jun  4 13:32 proc
dr-xr-x---.   4 root root  140 Jun  3 13:37 root
drwxr-xr-x.  45 root root 1160 Jun  4 13:33 run
lrwxrwxrwx.   1 root root    8 Aug  9  2021 sbin -> usr/sbin
drwxr-xr-x.   2 root root    6 Aug  9  2021 srv
dr-xr-xr-x.  13 root root    0 Jun  4 13:32 sys
drwxrwxrwt.  17 root root 4096 Jun  4 13:34 tmp
drwxr-xr-x.  12 root root  144 Jun  3 13:10 usr
drwxr-xr-x.  20 root root 4096 Jun  3 13:37 var
```

Example using `pwd` command:

```
[user@localhost ~]$ pwd
```

- The `pwd` command outputs your current directory location

Output:

```
/home/mmarin
```

Directory Listing Attributes

Type	# of Links	Owner	Group	Size	Month	Day	Time	Name
drwxr-xr-x.	21	root	root	4096	Feb	27	13:33	var
lrwxrwxrwx.	1	root	root	7	Feb	27	13:15	bin
-rw-r-r--	1	Root	Root	0	Mar	2	11:15	testfile

1st field (type)

- Any file that begins with "d" is a directory
- Any file that begins with "l" is a link
- Any file with nothing on it is a regular file

2nd field (number of links)

- The 2nd field tells you about the number of links it has or heart links that attach to the directory

3rd field (owner)

- The third field tells you the owner, who owns it.

4th field (group)

- The fourth field tells you the group of the directory. Which group owns that directory?

And so on with the next fields...

Creating Files and Directories

Creating Files (3 ways)

- `touch` - creates an empty file.
- `cp` - copying an existing file and creating a new file at the destination location
- `vi` - the vi editor command

Example using `touch` command:

```
[user@localhost ~]$ touch jerry
```

Example using `touch` command:

```
[user@localhost ~]$ touch kramer
```

- A name for the file must be stated in order to use the touch command to create a file

Example using `ls` command:

```
[user@localhost ~]$ ls -ltr
```

- `ls` shows the list of files and directories in the current location
- `-l` formats the list adding fields
- `t` makes the list ordered by time (newer on top)
- `r` reverses the list so that newer items appear on the bottom
- Is often used to verify that a certain file was created

Output:

```
total 0
drwxr-xr-x. 2 mmarin mmarin  6 Jun  3 13:42 Videos
drwxr-xr-x. 2 mmarin mmarin  6 Jun  3 13:42 Templates
drwxr-xr-x. 2 mmarin mmarin  6 Jun  3 13:42 Public
```

```
drwxr-xr-x. 2 mmarin mmarin  6 Jun  3 13:42 Pictures
drwxr-xr-x. 2 mmarin mmarin  6 Jun  3 13:42 Music
drwxr-xr-x. 2 mmarin mmarin  6 Jun  3 13:42 Desktop
drwxr-xr-x. 2 mmarin mmarin 77 Jun  4 14:26 Downloads
drwxr-xr-x. 3 mmarin mmarin 24 Jun  4 15:00 Documents
-rw-r--r--. 1 mmarin mmarin  0 Jun  4 15:31 jerry
-rw-r--r--. 1 mmarin mmarin  0 Jun  4 15:32 kramer
```

*Example using `cp` command:

```
[user@localhost ~]$ cp jerry lex
```

- `cp` is the command used to copy files and create a new one
- In this case the file 'jerry' is copied
- The new file created is named 'lex' and is a copy from the file 'jerry'

Example using `vi` command:

```
[user@localhost ~]$ vi homer
```

- The `vi` command creates a file named 'homer'

File editor:

```
~
~
~
~
:
```

- In contrast with `touch` and `cp` vi creates files and opens the file editor to start typing.
- if you want to exit the file editor in `vi` type the following:
  - `:wq!` - This is also referred as "bang"
  - While typing it, it will show up in the last line.

Example using `touch` command:

```
[user@localhost ~]$ touch bart lisa marge
```

- This single line creates three files using the `touch` command

Creating directories

- Creating directories
  - `mkdir`

Example using `mkdir` command:

```
[user@localhost ~]$ mkdir seinfield
```

- `mkdir` creates a directory in the current parent directory location
- A name for the directory has to be specified
- Only allowed when the user has sufficient rights to modify a directory
  - Usually not allowed in the `/` directory

Example using `mkdir` command:

```
[user@localhost ~]$ mkdir superman simpson
```

- `mkdir` also supports creating multiple directories in a single line

What happens if you do not have permission to modify a directory?

Example using `cd` command:

```
[user@localhost ~]$ cd /etc
```

- `cd` allows to change the current directory to another specified directory

Example using `touch` command to create a file inside a restricted directory:

```
[user@localhost etc]$ touch test123
```

Output:

```
touch: cannot touch 'test123': Permission denied
```

File Maintenance Commands

- `cp` - copies one file or directory to another
- `rm` - remove a file
- `mv` - move the location of a file from one to another OR to rename a file
- `mkdir` - make directory
- `rmdir` or `rm -r` - remove a directory
- `chgrp` - ownership of a file at the group level
- `chown` - ownership of a file at the user level

Copy

Current files shown by `ls -ltr`

```
total 0
drwxr-xr-x. 2 mmarin mmarin  6 Jun  3 13:42 Videos
drwxr-xr-x. 2 mmarin mmarin  6 Jun  3 13:42 Templates
drwxr-xr-x. 2 mmarin mmarin  6 Jun  3 13:42 Public
drwxr-xr-x. 2 mmarin mmarin  6 Jun  3 13:42 Pictures
drwxr-xr-x. 2 mmarin mmarin  6 Jun  3 13:42 Music
drwxr-xr-x. 2 mmarin mmarin  6 Jun  3 13:42 Desktop
drwxr-xr-x. 2 mmarin mmarin 77 Jun  4 14:26 Downloads
drwxr-xr-x. 3 mmarin mmarin 24 Jun  4 15:00 Documents
-rw-r--r--. 1 mmarin mmarin  0 Jun  4 15:31 jerry
-rw-r--r--. 1 mmarin mmarin  0 Jun  4 15:32 kramer
-rw-r--r--. 1 mmarin mmarin  0 Jun  4 15:39 george
-rw-r--r--. 1 mmarin mmarin  0 Jun  4 15:40 lex
-rw-r--r--. 1 mmarin mmarin  0 Jun  4 15:49 clark
-rw-r--r--. 1 mmarin mmarin  0 Jun  4 15:49 lois
-rw-r--r--. 1 mmarin mmarin  0 Jun  4 15:59 homer
-rw-r--r--. 1 mmarin mmarin  0 Jun  4 16:00 lisa
-rw-r--r--. 1 mmarin mmarin  0 Jun  4 16:00 bart
-rw-r--r--. 1 mmarin mmarin  0 Jun  4 16:00 marge
drwxr-xr-x. 2 mmarin mmarin  6 Jun  5 12:34 seinfeld
drwxr-xr-x. 2 mmarin mmarin  6 Jun  5 12:34 superman
drwxr-xr-x. 2 mmarin mmarin  6 Jun  5 12:35 simpson
echo
```

Example using `cp` command:

```
[user@localhost ~]$ cp george david
```

- Newest file created will be named "david"
- `cp` copied the file "george" and renamed "david" as a new file without removing the original "george" file

Example using `cp` command:

```
[user@localhost ~]$ cp david /tmp
```

- `cp` can be used to copy a certain file to a specified directory

---

Add file content and check file content

- `echo`
- `cat`

Example using `echo` command:

```
[user@localhost ~]$ echo "Hi my name is david" > david
```

- In this case `echo` is inserting the text line "Hi my name is david" to the specified file named "david"

Example using `cat` command:

```
[user@localhost ~]$ cat david
```

Output:

```
Hi my name is david
```

Remove

Example using `rm` command:

```
[user@localhost ~]$ rm apoho
```

- `rm` is removing a file named "apoho"
- The name of the file and directory has to be specified
- Remember `rm` only can remove files, not directories

Move

Example using `ls` command:

```
[user@localhost ~]$ ls -l lex
```

- This use of the `ls` command checks if a specified file, in this case named "lex", is currently in the location directory
- If the file exists in the current location it will output the fields of the file

Output:

```
-rw-r--r--. 1 mmarin mmarin 0 Jun  4 15:40 lex
```

Example using `mv` command:

```
[user@localhost ~]$ mv lex luther
```

- `mv` is moving the source file named "lex" to "luther"
- This is the renaming function of the `mv` command
  - Now the same "lex" file is named "luther"
  - It only changes the name, not the content

Example using `mv` command:

```
[user@localhost tmp]$ mv puddy /home/mmarin
```

- Note the current location is '/tmp'
- `mv` will move the file named "puddy" to the user home directory '/home/mmarin'
- This is the moving function of the `mv` command

Make directory

Example using `mkdir` command:

```
[user@localhost ~]$ mkdir gameofthrone
```

- Creates a directory named "gameofthrone" in the current location

Remove directory

Example using `rmdir` command:

```
[user@localhost ~]$ rmdir gameofthrone
```

- Removes the directory named "gameofthrone" from the current location

Example using `rm -r` command:

```
[user@localhost ~]$ rm -r gameofthrone
```

- Removes the directory named "gameofthrone" from the current location

Example using `rm -Rf` command:

```
[user@localhost ~]$ rm -Rf gameofthrone
```

- Removes the directory named "gameofthrone" from the current location
- `rm -Rf` will forcefully remove sub-directories and its contents as well

Change ownership

Become root

- Use `whoami` command to find out who we are
- Use the `su -` command
- Type the Admin password

Example using `whoami` command:

```
[mmarin@localhost ~]$ whoami
```

- Returns the current user

Output:

```
mamarin
```

Example using `su -` command:

```
[user@localhost ~]$ su -
```

Output:

```
Password:
[root@localhost ~]#
```

- Note how the prompt changed
  - Now we are root
  - Now instead of ending with a "$" it is a "#"
- The root user provides all the permissions you can think of.

Example using `cd` command:

```
[root@localhost ~]# cd /home/mmarin
```

- Travel to your user location using the `cd` command

Example using `chgrp` command:

```
[root@localhost ~]# chgrp root puddy
```

- `chgrp` changes the ownership at level group from the file named "puddy" to 'root'
- Note the order of this command is the following:
  - `chgrp [new ownership name] [file name]`
  - Also works with directories

Output from `ls -ltr puddy`

```
-rw-r--r--. 1 mmarin root 19 Jun  5 12:57 puddy
```

Example using `chown` command:

```
[root@localhost ~]# chown root puddy
```

- `chown` changes the ownership at level user from the file named "puddy" to 'root'
- - Note the order of this command is the following:
  - `chown [new ownership name] [file name]`
  - Also works with directories

Output from `ls -ltr puddy`

```
-rw-r--r--. 1 root root 19 Jun  5 12:57 puddy
```

Example using `chown` command:

```
[root@localhost ~]# chown mmarin:mmarin puddy
```

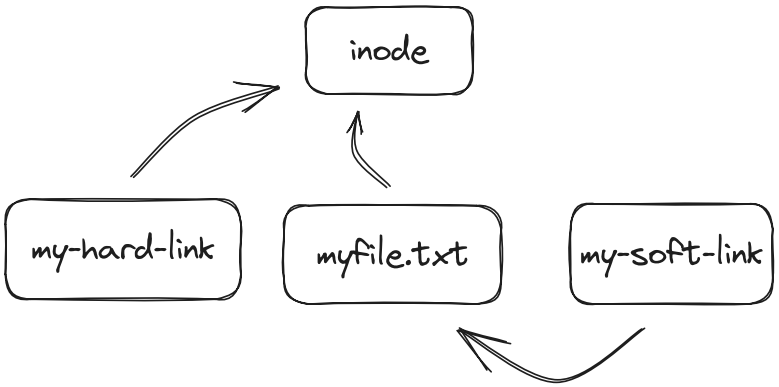
- In this instance of the `chown` command it sets the ownership from both user and group level to the user 'mmarin'

Output from `ls -ltr puddy`

```
-rw-r--r--. 1 mmarin mmarin 19 Jun  5 12:57 puddy
```

Soft and Hard Links

- A link is like a shortcut for convenience access to certain files
  - Can be seen as creating a shortcut in the Windows desktop to a certain file for much easier access
- **inode** = Pointer or number of a file on the hard disk
  - Computer does not understand file names or any name
  - Computers understand numbers
  - Every time you create a file, the computer assigns a number to it on the hard disk and associate that number to it
  - Than number is called inode
  - Every time you try to read or retrieve that file it goes to that number
- **Soft Link** = Link will be removed if file is removed or renamed
  - If you create a file and you create a soft link to that file. So onece it is created, it actually looks for that inode through that file so that creates a soft link
  - If you remove the source file, it will remove the destination link
- **Hard Link** = Deleting renaming or moving the original file will not affect the hard link
- Commands to create a Hard and Soft links:
  - `ln` - Create a hard link
  - `ln -s` - Create a soft link



Steps to create a link in the '/tmp' location from a file in the home directory:

1. Create a file in your home directory
   - `[user@localhost ~]$ thouch hulk` - creates a file named touch
2. Change to destination directory
   - `[user@localhost ~]$ cd /tmp` - travels to the '/tmp' directory
3. Create link:

Example using `ln -s` command:

```
[user@localhost tmp]$ ln -s /home/mmarin/hulk
```

- Creates a soft link from the file 'hulk', retrieved from the home folder of the "mmarin" user, to the '/tmp' directory

Output from `ls -ltr` :

```
...
-rw-r--r--. 1 mmarin mmarin     19 Jun  5 12:58 david
lrwxrwxrwx. 1 mmarin mmarin     17 Jun  5 14:37 hulk -> /home/mmarin/hulkls
```

- Remember links start with "l"
- You can use `echo` to insert file contents in the original file and if accessed from the link using `cat` it will show the same contents from the original.

Example using `ls -li` command:

```
[user@localhost ~]$ ls -li
```

- The `i` specifier of the `ls` command stands for inode. It will show the longlisting format and the inode number for each file and directory

Output:

```
52155496 drwxr-xr-x. 2 mmarin mmarin  6 Jun  5 12:34 superman
52142028 drwxr-xr-x. 2 mmarin mmarin  6 Jun  3 13:42 Templates
```

```
2004497 drwxr-xr-x. 2 root   root    6 Jun  5 13:49 uwu
1578706 drwxr-xr-x. 2 mmarin mmarin  6 Jun  3 13:42 Videos
```

- The inode number at the left is the number associated with that specific file.
- This is the number that is remembered by your Operative System, not the name itself.
- Each inode will be different from another. Even when links are present within the directory. A link will have a different inode from that of the file it is linked to. This is only true when creating Soft Links
- If a Soft Link is removed from a directory the link will still appear when using the `ls -ltr` command but it will be highlited in red meaning that it is no longer usable
  - When trying to use the `cat` command to retrieve contents from a link to a source file that was deleted the terminal will output `cat: hulk: No such file or directory`. In this case a source file named 'hulk' was previously removed and when attempting to use a link in a different directory it won't work.
  - This only applies to Soft Links
  - So it is better to just remove the link as well
- If a Hard Link is removed from a directory the link will still appear when using the `ls -ltr` command and it will still be usable.
  - Please note that when creating a Hard Link the inode attached to the file is just copied to the directory where the link is created so bothe the link and the actual file will have the same inode number.
  - In other words, there is no pointing to other file.
  - When removing the source file, the link with the same inode number will still be reachable, in other words, it does not remove the destination.
  - If you get this error: `ln: failed to create hard link './hulk' => '/home/mmarin/hulk': Invalid cross-device link`
    - It is most likely that your '/home' directory isn't on the same partition as the '/tmp' directory
    - Hard Links only work within the same partition

Example using `echo` command:

```
[user@localhost ~]$ echo "123" >> hulk
```

- If you want to add new content in a new line without overwriting the file you can use the `>>` operand instead of just `>`.
- `>` Overwrites or erases the content and adds the new one to the file.
- `>>` Adds a new line and adds the content there without erasing the already existing content in the file.
- As we have a link in the '/tmp' directory and we are modifying this file in its original location, changes will also show up when using the link in the '/tmp' location.

Output from `cat hulk` :

```
hulk is a superhero
123
```

Input and Output Redirects

There are 3 redirects in Linux

1. Standard input (`stdin`) and it has file descriptor number as 0
2. Standard output (`stdout`) and it has file descriptor number as 1
3. Standard error (`sterr`) and it has file descriptor number as 2

Symbol	Description
>	Directs the standard output of a command to a file. If the file exists, it is overwritten.
>>	Directs the output to a file, adding the output to the end of the existing file.
2>	Directs standard error to the file.
2>>	Directs the standard error to a file, adding the output to the end of the existing file.
&>	Directs standard output and standard error to the file.
<	Directs the contents of a file to the command.
<<	Accepts text on the following lines as standard input.
<>	The specified file is used for both standard input and standard output.

- Please consider that everything in Linux is considered as a File
  - This means that any peripherals like mouse, keyboard or monitor are also seen by the OS as a file.
- When we actually write something in the keyboard that is coming in as a standard input and it is actually knocking on the door for file zero, file descriptor 0.
- When we are showing that output on the screen that output is actually going thourgh the output descriptor number 1.
- if there's any error it will show up as the file descriptor 2.

Output (`stdout`) - 1

- By default when running a command its output goes to the terminal
- The output of a command can be routed to a file using `>` symbol
  - If you are typing a command you could route the output that you see from the command to another file.
  - E.g. `ls -l > listings`
    - In this example the outputs of the `ls -l` command are being routed or in other words being saved into the file with the name 'listings'
  - E.g. `pwd > findpath`
- If using the same file for additional output or to append to the same file then use `>>`
  - E.g. `ls -la >> listings`
  - E.g. `echo "Hello World" >> findpath.`

Example using `ls -la` command:

```
[user@localhost ~]$ ls -la >> listings
```

- The `a` specifier of the `ls` command allows to show all the hidden files of a directory
- In this line we are routing the output from the `ls -la` command and appending it to the file named 'listings' using the `>>` redirect
- You can use `cat` to see the contents of this file

Input (`stdin`) - 0

- Input is used when feeding file contents to a file
  - E.g. `cat < listings`
  - E.g. `mail -s "Office memo" allusers@abc.com < memoletter`

- This line sends a mail with the subject line "Office memo" and it is being sent to all the users of the company abc.com.
- Now what is going to be the content of that letter?
  - The content of that letter is actually written inside the 'memoletter' file

Example using `cat` command:

```
[user@localhost ~]$ cat < findpath
```

- In common language we could read this line as "cat input findpath"
- As cat itself is a command to show the contents of a file it is the same to do `cat findpath`
- In this case we are explicitly telling the `cat` command to use the feedings from the 'findpath' file

Output: (contents found inside the 'findpath' file)

```
/home/mmarin
Hellow World
```

Error (`stderr`) - 2

- When a command is executed we use the keyboard and that is also considered (stdin - 0)
- That command output goes on the monitor and that output is (stdout - 1)
- If the command produces an error on the screen then it is considered (stderr - 2)
  - We can use redirects to route errors from the screen to a different file
    - E.g. `ls -l /root 2> errorfile`
      - In this case we know that '/root' is owned by root, not by a regular use, it will produce an error.
      - We could route that to a file, but we have to specify how we are routing
      - We only want to route the standard error, not the entire output. and that is why we have to use the `2>` redirect sign.
    - E.g. `telnet localhost 2> errorfile`
      - This is another example of one use of the `2>` redirect
      - We could use a program like Telnet, Localhost and its error message if it's not gonna be able to connect to that host, we do not want to see that on the screen error, we just want to go into the 'errorfile'

Example using `telnet` command:

```
[user@localhost ~]$ telnet localhost
```

- Telnet is a small program which actually allows you to connect from one machine to another just like SSH.
- 'localhost' is your own host

Output:

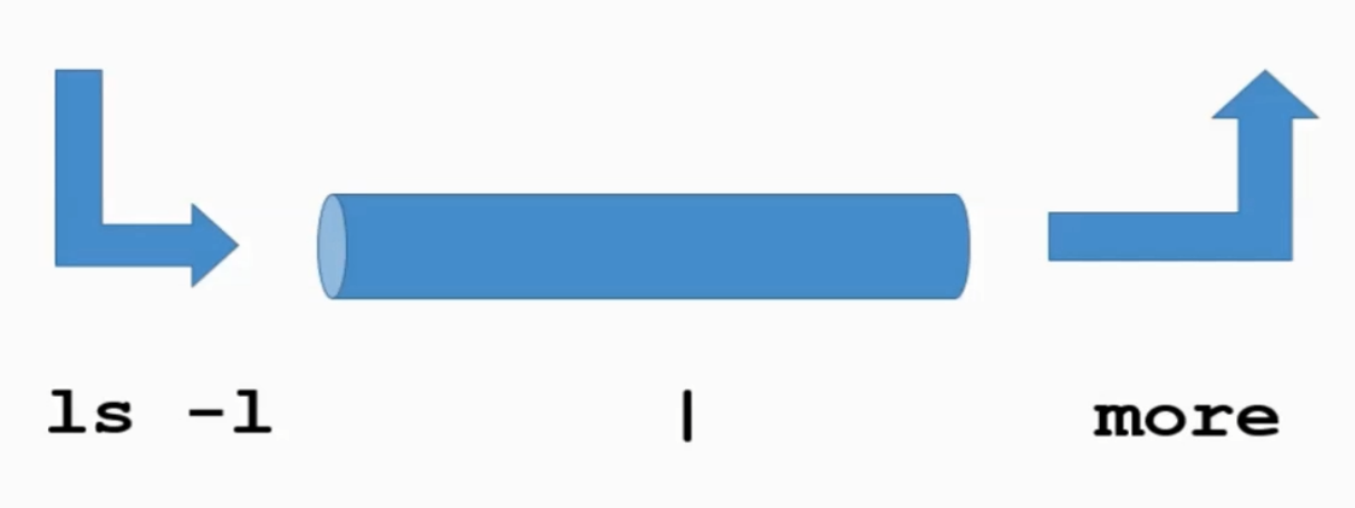
```
Trying ::1...
telnet: connect to address ::1: Connection refused
Trying 127.0.0.1...
telnet: connect to address 127.0.0.1: Connection refused
```

- We know this like will throw an standard error that we can route into a file using the following line: `telnet localhost 2> errorfile`
  - In that case we are routing the error thrown by the `telnet` command which is actually only the last 2 lines that say "Connection refused" to the file named 'errorfile'
  - Note we can also use `2>>` to append a routed standard error to a file without overwriting the whole file

> Input and Output redirects are very useful when you have to create shell scripts and automating tasks

Pipes (`|`)

- A pipe is used by the shell to connect the output of one command directly to the input of another command.
  - The symbol for a pipe is the vertical bar (`|`). The command syntax is:
    - `command1 [arguments] | command2 [arguments]`
- In you want to take the output of the first command and pipe it to a different output then you could use a pipe.



- A command goes into a pipe and the output of that command will then be refined by the last command.

Case example: If you want to view the contents of the `ls -ltr` command of the '\etc' directory (this directory contains a lot of files) one page at a time you can do the following:

Example using `more` command:

```
[user@localhost etc]$ ls -ltr | more
```

- The `more` command gives you the output of a file one page at a time.
- An `ls -ltr` command is being piped with a `more` command to show the outputs of the `ls -ltr` one page at a time

Output:

```
...
drwxr-xr-x.  2 root root        6 Mar 23  2022 cron.weekly
-rw-r--r--.  1 root root      451 Mar 23  2022 crontab
drwxr-xr-x.  2 root root        6 Mar 23  2022 cron.monthly
--More--
```

- Note that at the bottom it tells you that there is more.

- You can hit space bar to go down to the next page will show up until the file ends.

Example using `ll` command:

```
[user@localhost ~]$ ll
```

- `ll` Is an easier way of typing `ls -l` (It is the same)

Output:

```
...
drwxr-xr-x.  2 root root         6 Mar 23  2022 cron.weekly
-rw-r--r--.  1 root root       451 Mar 23  2022 crontab
drwxr-xr-x.  2 root root         6 Mar 23  2022 cron.monthly
```

Case example 2: What if you wanted to get the last line of your output? Well you can do the following:

Example using `tail` command:

```
[user@localhost ~]$ ll | tail -1
```

- `tail` gives you the last lines of your output
- The `-1` specifier tells the `tail` command to output exactly only the last line of the output. Without this specifier it will throw multiple lines that are considered as the "tail" of the output.
- In this case the `ll` command is being piped with the `tail -1` command to show only the last line of the output

Output:

```
drwxr-xr-x.  2 root root        25 Jun  3 13:24 yum.repos.d
```