**Find**

[ Create a user ]

# useradd sainath

[ changing the pwd ]

# cd /home/sainath

[ create a file ]

#touch file.txt

Add data into the file using cat or vi

[ Find linux files by name or extension ]

# find /home/sainath/ -name “\*.txt”

[ Find a file called testfile.txt in current ]

# find . -name file.txt

[ Create a file ‘raj.tar’ ]

# touch raj.tar

[ Find all .tar files in the /home and sub-directories ]

# find /home -name \*.tar

[ Find an empty file within the current directory ]

# find . -empty -type f

[ to find the files modified in last one day]

# find /home/raj -mtime -1

[ to find files modified ending with “.txt”in last 7 days]

# find /home/raj -name "\*.txt" -mtime -7

[ Find files by modification time ]

# find / -name "\*.txt" -mtime -7

find -name <filename.txt>

find -iname <filename.txt>

for more info, <http://www.binarytides.com/linux-find-command-examples/>

**Archiving files and compressing archives**

[ create a directory]

# mkdir rajkumar

# cd rajkumar

[ creating multiple files at a time ]

# touch file{1..10}

# ls

[ To compress an entire ‘test’ directory along with files in it and zipping ]

# tar czvf rajkumar.tar.gz ./rajkumar

[ to list contents in archive file]

# tar tvf ./rajkumar.tar.gz

[create two directories with files in it]

# mkdir ram laxman

# cd ram

# touch file{1..10}

# cd ..

# cd laxman

# touch file{1..10}

# cd ..

# ls

[ to compress multiple directories along with data into one archive and zipping ]

# tar czvf ramlaxman.tar.gz ./laxman ./laxman

[ to remove a directory with files in it]

# rm –r ram

# rm –r laxman

[ to extract the archive]

# tar xvfz ramlaxman.tar.gz

[ to remove archive file ]

# rm ramlaxman.tar.gz

[ Archiving excluding directories or files in a directory]

# tar czvf ramlaxman.tar.gz ./ram --exclude=./ram/file1

[ to list the contents of archive file ]

# tar tvf ramlaxman.tar.gz

[ to remove tar file]

# rm ramlaxman.tar.gz

[ to compress a directory]

# tar xvf ramlaxman.tar

[ gzip a compressed file]

# gzip ramlaxman.tar

#ls

[gunzip a zip file]

# gunzip ramlaxman.tar.gz

#ls

[ bzip2 a compressed file]

# bzip2 ramlaxman.tar

#ls

[ bunzip2 a zip file]

# bunzip2 ramlaxman.tar.gz

#ls

For more info, <http://www.tecmint.com/18-tar-command-examples-in-linux/>

**Word count**

The wc (word count) command in Unix/Linux operating systems is used to find out number of newline count, word count, byte and characters count in a files specified by the file arguments. The syntax of wc command as shown below.

# wc <options> <filename>

wc -l : Prints the number of lines in a file.

wc -w : prints the number of words in a file.

wc -c : Displays the count of bytes in a file.

wc -m : prints the count of characters from a file.

wc -L : prints only the length of the longest line in a file.

[ create a file]

# touch raj

[ Append data into file ‘raj’ ]

# cat >> raj

Hi

Hello

Welcome to party

Save the file(ctrl+d)

[ to check no of lines, no of words, no of bytes ]

# wc raj

[ to check no of lines ]

# wc –l raj

[ to check no of words]

# wc –w raj

[ to check no of bytes or characters]

# wc –c raj

# wc –m raj

[ to display length of longest line ]

# wc –L raj

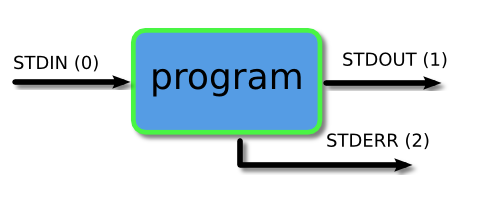
For more info, <http://www.tecmint.com/wc-command-examples/>

**Piping and Redirection**

STDIN (0) - Standard input (data fed into the program)

STDOUT (1) - Standard output (data printed by the program, defaults to the terminal)

STDERR (2) - Standard error (for error messages, also defaults to the terminal)



**Redirecting to a file**

[ create few files]

# touch file{1..10}

# ls

[ sending the output of ‘ls’ into a file ‘myoutput’ ]

# ls > myoutput

[ verify for the file ‘myoutput’ ]

# ls

[ check the contents of file ‘myoutput’ ]

# cat myoutput

**Saving to an existing file**

[ Appending data into ‘file1’ ]

# cat >> file1

Hello raj

Press (ctrl+d) to save

[word counting in file1 and adding the output to file ‘myoutput’ ]

# wc -l file1 > myoutput

[ check the content of ‘myoutput’ file ]

# cat myoutput

[ We can instead get the new data to be appended to the file by using the double greater than operator ( >> ) ]

# ls >> myoutput

# cat myoutput

**Redirecting from a file**

If we use the less than operator ( < ) then we can send data the other way. We will read data from the file and feed it into the program via it's STDIN stream.

# wc -l myoutput

# wc -l < myoutput

A lot of programs (as we've seen in previous sections) allow us to supply a file as a command line argument and it will read and process the contents of that file. Given this, you may be asking why we would need to use this operator. The above example illustrates a subtle but useful difference. You'll notice that when we ran wc supplying the file to process as a command line argument, the output from the program included the name of the file that was processed. When we ran it redirecting the contents of the file into wc the file name was not printed. This is because whenever we use redirection or piping, the data is sent anonymously. So in the above example, wc recieved some content to process, but it has no knowledge of where it came from so it may not print this information. As a result, this mechanism is often used in order to get ancillary data (which may not be required) to not be printed.

We may easily combine the two forms of redirection we have seen so far into a single command as seen in the example below.

# wc -l < raj> myoutput

# cat myoutput

**Redirecting STDERR**

**Stream 0: STDIN**

**Stream 1: STDOUT**

**Stream 2: STDERR**

Now let's look at the third stream which is Standard Error or STDERR. The three streams actually have numbers associated with them. STDERR is stream number 2 and we may use these numbers to identify the streams. If we place a number before the > operator then it will redirect that stream (if we don't use a number, like we have been doing so far, then it defaults to stream 1).

[ create a file ‘raj’ ]

# touch raj

# ls -l raj sai

**Output:**

ls: cannot access sai: No such file or directory

-rw-r--r--. 1 root root 0 Apr 12 05:04 raj

[ to add the errors in the output to a file]

# ls -l raj sai 2> file

**Output:**

-rw-r--r--. 1 root root 0 Apr 12 05:04 raj

# cat file

**Output:**

ls: cannot access sai: No such file or directory

May be we wish to save both normal output and error messages into a single file. This can be done by redirecting the STDERR stream to the STDOUT stream and redirecting STDOUT to a file. We redirect to a file first then redirect the error stream. We identify the redirection to a stream by placing an & in front of the stream number (otherwise it would redirect to a file called 1).

# ls -l raj sai > myoutput 2>&1

# cat myoutput

ls: cannot access sai: No such file or directory

-rw-r--r--. 1 root root 0 Apr 12 05:04 raj

**Piping**

So far we've dealt with sending data to and from files. Now we'll take a look at a mechanism for sending data from one program to another. It's called piping and the operator we use is ( | ) (found above the backslash ( \ ) key on most keyboards). What this operator does is, feed the output from the program on the left as input to the program on the right. In the example below we will list only the first 3 files in the directory.

[ create a directory ]

# mkdir rajkumar

# cd rajkumar

[ create 5 files]

# touch file{1..5}

# ls

[ listing only first three files using piping ]

# ls | head -3

We may pipe as many programs together as we like. In the below example we have then piped the output to tail so as to get only the third file.

# ls | head -3 | tail -1

You may combine pipes and redirection too.

# ls | head -3 | tail -1 > myoutput

# cat myoutput

**Summary of Redirecting and piping**

>

Save output to a file.

>>

Append output to a file.

<

Read input from a file.

2>

Redirect error messages.

|

Send the output from one program as input to another program.