Linux Commands Part 3

This is a continuation from my second Linux commands tutorial (Part #2), where I introduced you to file permissions, input/output redirection and the command pipeline. It can be accessed **here**.

To complete this tutorial, you will need access to a running Linux distribution, or 'distro' for short. There are a number of Linux 'distros'. If you do not already have access to a Linux system, I have a number of VirtualBox tutorials where I demonstrate the installation of CentOS 7 and Ubuntu 22 as virtual machines, accessible **here**.

I also have a few tutorials where I demonstrate the creation of AWS compute instances, RHEL 8 and Ubuntu 20, accessible **here**. Keep in mind that you will need to create an AWS account to be able to create a compute instance. If you do not have an AWS account, my tutorial **Create AWS Free Tier Account** is accessible **here.**

if you prefer, you can use a CentOS 7 or Ubuntu 22 VM instead. The choice is yours.

For this tutorial, I will be using both my RHEL 8, and Ubuntu 20, AWS compute instances, and I will be providing screenshots from my Ubuntu 20 instance. When I encounter a difference in output, I will include the RHEL 8 screenshot.

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

Below is a brief listing of the commands and features 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
`find`	• find files on the system
`history`	• list previously executed commands
`grep`	• print lines that match patterns
`tr, tee, wc, cut`	• Miscellaneous Text Utilities
`sort, uniq, paste, join`	• File Manipulation Utilities

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

Please note, if you completed **Linux Basic Commands Part 2**, you can proceed to **Wildcard Operators**.

If not, execute the following commands to ensure that our command outputs, throughout the tutorial, match.

```
$ echo "This is test file #1." > test_file1
$ echo "This is the second line." >> test_file1
$ chown root:root test_file1
$ echo "This is test file #3." > test_file4
$ echo "echo 'test script #1.'" > abc.sh
$ echo "echo 'test script #2.'" > def.sh
$ chmod +x abc.sh
$ chmod +x def.sh
$ mkdir -p /home/ubuntu/projects/my_linux_stuff
```

Wildcard Operators

You can search for a filename containing specific characters using wildcards.

Wildcard	Result
?	Matches any single character
*	Matches any string of characters (0 or more characters)
[set]	Matches any character in the set of characters, for example [ehz] will match any occurrence of "e", "h", or "z"
[!set]	Matches any character not in the set of characters

We will create a few empty files to work with.

$ touch bat.out test.out abcd.sh test1.log test2.log

$ ls -ltr

```
ubuntu@ip-172-31-4-185:~$ touch bat.out test.out abcd.sh test1.log test2.log
ubuntu@ip-172-31-4-185:~$
ubuntu@ip-172-31-4-185:~$ ls -ltr
total 20
drwxrwxr-x 3 ubuntu ubuntu 4096 Apr 18 10:18 projects
-rw-rw-r-- 1 ubuntu ubuntu   21 Apr 21 14:28 test_file4
-rw-rw-r-- 1 root   root     72 Apr 21 14:39 test_file1
-rwxrwxr-x 1 ubuntu ubuntu   23 Apr 21 14:39 def.sh
-rwxrwxr-x 1 ubuntu ubuntu   23 Apr 21 14:39 abc.sh
-rw-rw-r-- 1 ubuntu ubuntu    0 Apr 21 14:40 test2.log
-rw-rw-r-- 1 ubuntu ubuntu    0 Apr 21 14:40 test1.log
-rw-rw-r-- 1 ubuntu ubuntu    0 Apr 21 14:40 test.out
-rw-rw-r-- 1 ubuntu ubuntu    0 Apr 21 14:40 bat.out
-rw-rw-r-- 1 ubuntu ubuntu    0 Apr 21 14:40 abcd.sh
ubuntu@ip-172-31-4-185:~$
```

Now, execute the following commands and observe the results.

$ ls -l ba?.out

$ ls -l ab?.sh

$ ls -l abc*.sh

$ ls -l test?.log

$ ls -l t???.out

$ ls -l *.sh

$ ls -l [ab]*

$ ls -l [!lpt]*

```
ubuntu@ip-172-31-4-185:~$ ls -l ba?.out
-rw-rw-r-- 1 ubuntu ubuntu 0 Apr 21 14:40 bat.out
ubuntu@ip-172-31-4-185:~$ ls -l ab?.sh
-rwxrwxr-x 1 ubuntu ubuntu 23 Apr 21 14:39 abc.sh
ubuntu@ip-172-31-4-185:~$ ls -l abc*.sh
-rwxrwxr-x 1 ubuntu ubuntu 23 Apr 21 14:39 abc.sh
-rw-rw-r-- 1 ubuntu ubuntu  0 Apr 21 14:40 abcd.sh
ubuntu@ip-172-31-4-185:~$ ls -l test?.log
-rw-rw-r-- 1 ubuntu ubuntu 0 Apr 21 14:40 test1.log
-rw-rw-r-- 1 ubuntu ubuntu 0 Apr 21 14:40 test2.log
ubuntu@ip-172-31-4-185:~$ ls -l t???.out
-rw-rw-r-- 1 ubuntu ubuntu 0 Apr 21 14:40 test.out
ubuntu@ip-172-31-4-185:~$ ls -l *.sh
-rwxrwxr-x 1 ubuntu ubuntu 23 Apr 21 14:39 abc.sh
-rw-rw-r-- 1 ubuntu ubuntu  0 Apr 21 14:40 abcd.sh
-rwxrwxr-x 1 ubuntu ubuntu 23 Apr 21 14:39 def.sh
ubuntu@ip-172-31-4-185:~$ ls -l [ab]*
-rwxrwxr-x 1 ubuntu ubuntu 23 Apr 21 14:39 abc.sh
-rw-rw-r-- 1 ubuntu ubuntu  0 Apr 21 14:40 abcd.sh
-rw-rw-r-- 1 ubuntu ubuntu  0 Apr 21 14:40 bat.out
ubuntu@ip-172-31-4-185:~$ ls -l [!lpt]*
-rwxrwxr-x 1 ubuntu ubuntu 23 Apr 21 14:39 abc.sh
-rw-rw-r-- 1 ubuntu ubuntu  0 Apr 21 14:40 abcd.sh
-rw-rw-r-- 1 ubuntu ubuntu  0 Apr 21 14:40 bat.out
-rwxrwxr-x 1 ubuntu ubuntu 23 Apr 21 14:39 def.sh
ubuntu@ip-172-31-4-185:~$
```

To search for files using the ? wildcard, replace each unknown character with ?.

To search for files using the * wildcard, replace the unknown string with *. Note that the * wildcard refers to zero, or more, characters, as demonstrated with the '**ls -l abc*.sh**' command above.

Using the square brackets for searches allows you to specify a set of characters at a given position. For example, the command ls -l [ab]* returns all filenames whose 1st character is either a or b.

Search for Files using find

When no arguments are given, find lists all files in the current directory and all of its subdirectories.

$ find

```
ubuntu@ip-172-31-4-185:~$ find
.
./.bashrc
./test.out
./.ssh
./.ssh/authorized_keys
./test_file1
./.cache
./.cache/motd.legal-displayed
./bat.out
./.viminfo
./.Xauthority
./projects
./projects/my_linux_stuff
./.bash_logout
./.bash_aliases
./.sudo_as_admin_successful
./test1.log
./def.sh
./test2.log
./.bash_history
./test_file4
./.profile
./abc.sh
./.lesshst
./abcd.sh
ubuntu@ip-172-31-4-185:~$
```

Commonly used options to shorten the list include **-name** (only list files with a certain pattern in their name), **-iname** (also ignore the case of file names), and **-type** (which will restrict the results to files of a certain specified type, such as **d** for directory, **l** for symbolic link or **f** for a regular file).

Searching for files and directories named "**gcc**":

$ find /usr -name gcc

```
ubuntu@ip-172-31-4-185:~$ find /usr -name gcc
/usr/src/linux-headers-5.11.0-1022-aws/include/config/gcc
/usr/src/linux-headers-5.11.0-1022-aws/include/config/have/gcc
/usr/src/linux-aws-5.13-headers-5.13.0-1021/scripts/dummy-tools/gcc
/usr/src/linux-aws-5.13-headers-5.13.0-1019/scripts/dummy-tools/gcc
/usr/src/linux-aws-5.11-headers-5.11.0-1022/scripts/dummy-tools/gcc
/usr/share/bash-completion/completions/gcc
/usr/share/gcc
ubuntu@ip-172-31-4-185:~$
```

Searching only for directories named "**gcc**":

$ find /usr -type d -name gcc

```
ubuntu@ip-172-31-4-185:~$ find /usr -type d -name gcc
/usr/src/linux-headers-5.11.0-1022-aws/include/config/gcc
/usr/src/linux-headers-5.11.0-1022-aws/include/config/have/gcc
/usr/share/gcc
ubuntu@ip-172-31-4-185:~$
```

Searching for files, in the current directory, whose names include "test":

$ find . -type f -name "test*"

```
ubuntu@ip-172-31-4-185:~$ find . -type f -name "test*"
./test.out
./test_file1
./test1.log
./test2.log
./test_file4
ubuntu@ip-172-31-4-185:~$
```

We can also find files based on when they were created **(-ctime**), last accessed **(-atime**) and modified **(-mtime**). Each option takes a number of days and can be expressed as either a number (**n**) that means exactly that value, **+n** which means greater than that number, or **-n** which means less than that number. There are similar options for times in minutes (as in **-cmin**, **-amin**, and **-mmin**).

Checking log files is a good practice when troubleshooting system issues, we will use the **/var/log** directory as our search directory for the following commands. Please note that we will be using **sudo** to avoid "**Permission denied**", for files we do not have access to.

```
# files created over 10 days ago
```

```
$ sudo find /var/log -ctime +10
```

```
ubuntu@ip-172-31-4-185:~$ sudo find /var/log -ctime +10
/var/log/unattended-upgrades
/var/log/unattended-upgrades/unattended-upgrades-shutdown.log
/var/log/dist-upgrade
/var/log/journal
/var/log/journal/42933177b5a74b2e871bd6e0d07f4974
/var/log/private
/var/log/landscape
/var/log/landscape/sysinfo.log
/var/log/amazon
/var/log/amazon/ssm
/var/log/amazon/ssm/audits/amazon-ssm-agent-audit-2022-03-30
ubuntu@ip-172-31-4-185:~$
```

```
# files accessed 1 day ago
```

```
$ sudo find /var/log -atime 1
```

```
ubuntu@ip-172-31-4-185:~$ sudo find /var/log -atime 1
/var/log/journal/42933177b5a74b2e871bd6e0d07f4974/system.journal
/var/log/syslog.1
ubuntu@ip-172-31-4-185:~$
```

```
# files modified less than 1 day ago
```

```
$ sudo find /var/log -mtime -1
```

```
ubuntu@ip-172-31-4-185:~$ sudo find /var/log -mtime -1
/var/log
/var/log/syslog
/var/log/unattended-upgrades/unattended-upgrades-dpkg.log
/var/log/unattended-upgrades/unattended-upgrades.log
/var/log/alternatives.log
/var/log/journal/42933177b5a74b2e871bd6e0d07f4974/user-1000.journal
/var/log/journal/42933177b5a74b2e871bd6e0d07f4974/system.journal
/var/log/apt
/var/log/apt/history.log
/var/log/apt/eipp.log.xz
/var/log/apt/term.log
/var/log/syslog.1
/var/log/ubuntu-advantage-timer.log
/var/log/btmp
/var/log/auth.log
/var/log/wtmp
/var/log/lastlog
/var/log/dpkg.log
ubuntu@ip-172-31-4-185:~$
```

```
# files modified less that 60 mins ago
```

```
$ sudo find /var/log -amin -60
```

```
ubuntu@ip-172-31-4-185:~$ sudo find /var/log -amin -60
/var/log
/var/log/unattended-upgrades
/var/log/dist-upgrade
/var/log/journal
/var/log/apt
/var/log/private
/var/log/landscape
/var/log/amazon
/var/log/amazon/ssm
/var/log/amazon/ssm/audits
ubuntu@ip-172-31-4-185:~$
```

We can also find files based on size.

```
$ find . -size 0
```

```
ubuntu@ip-172-31-4-185:~$ find . -size 0
./test.out
./.cache/motd.legal-displayed
./bat.out
./.sudo_as_admin_successful
./test1.log
./test2.log
./abcd.sh
ubuntu@ip-172-31-4-185:~$
```

You will notice that some of the empty files we created earlier were included in the result.

Note the size here is in 512-byte blocks, by default; you can also specify bytes (c), kilobytes (k), megabytes (M), gigabytes (G), etc.

As with the time numbers above, file sizes can also be exact numbers (n), +n or -n.

We will now check the file sizes of the /boot directory

```
$ find /boot -size +10M

$ find /boot -size +30M -size -50M
```

```
ubuntu@ip-172-31-4-185:~$ find /boot -size +10M
/boot/initrd.img-5.11.0-1022-aws
/boot/initrd.img-5.13.0-1021-aws
/boot/initrd.img-5.13.0-1019-aws
/boot/vmlinuz-5.11.0-1022-aws
ubuntu@ip-172-31-4-185:~$ find /boot -size +30M -size -50M
/boot/initrd.img-5.11.0-1022-aws
/boot/initrd.img-5.13.0-1021-aws
/boot/initrd.img-5.13.0-1019-aws
ubuntu@ip-172-31-4-185:~$
```

Another good use of find is being able to run commands on the files that match your search criteria. The **-exec** option is used for this purpose.

In the current directory, find and remove all files that end with .log:

```
$ ls -l *.log

$ find . -name "*.log" -exec rm {} \;

$ ls -l *.log
```

```
ubuntu@ip-172-31-4-185:~$ ls -l *.log
-rw-rw-r-- 1 ubuntu ubuntu 0 Apr 21 14:40 test1.log
-rw-rw-r-- 1 ubuntu ubuntu 0 Apr 21 14:40 test2.log
ubuntu@ip-172-31-4-185:~$
ubuntu@ip-172-31-4-185:~$ find . -name "*.log" -exec rm {} \;
ubuntu@ip-172-31-4-185:~$
ubuntu@ip-172-31-4-185:~$ ls -l *.log
ls: cannot access '*.log': No such file or directory
ubuntu@ip-172-31-4-185:~$
```

You will notice that both of our files, test1.log & test2.log, were deleted.

The {} (squiggly brackets) will contain all the file names returned from the find search, and the preceding command (**rm**) will be run on each file. There must be a space after the {} and the ; must be escaped, **\;** , for the command to run.

Another example would be to get a full directory listing of the files returned.

```
$ find /boot -size +20 -size -50M -exec ls -l {} \;
```

```
ubuntu@ip-172-31-4-185:~$ find /boot -size +10M -size -50M -exec ls -l {} \;
-rw-r--r-- 1 root root 35466202 Nov 29 23:37 /boot/initrd.img-5.11.0-1022-aws
-rw-r--r-- 1 root root 35957607 Apr 14 06:09 /boot/initrd.img-5.13.0-1021-aws
-rw-r--r-- 1 root root 35953091 Apr 13 11:54 /boot/initrd.img-5.13.0-1019-aws
-rw------- 1 root root 14750208 Nov 15 13:20 /boot/vmlinuz-5.11.0-1022-aws
ubuntu@ip-172-31-4-185:~$
```

The find command has many options. For details consult either the find command's --help option or it’s man page.

Recalling Previous Commands

bash keeps track of previously entered commands and statements in a history buffer; you can recall previously used commands simply by using the Up and Down cursor keys. To view the list of previously executed commands, you can just type history at the command line. (**NOTE**: I did not provide a screenshot due to the size of my file)

```
$ history
```

The list of commands is displayed with the most recent command appearing last in the list. This information is stored in **~/.bash_history**. (**NOTE**: the tilde **~** represents your home directory, /home/ubuntu/.bash_history)

Several associated environment variables can be used to get information about the history file.

HISTFILE	stores the location of the history file
HISTFILESIZE	stores the maximum number of lines in the history file
HISTSIZE	stores the maximum number of lines in the history file for the current session

```
$ echo $HISTFILE
$ echo $HISTFILESIZE
$ echo $HISTSIZE
```

```
ubuntu@ip-172-31-4-185:~$ echo $HISTFILE
/home/ubuntu/.bash_history
ubuntu@ip-172-31-4-185:~$ echo $HISTFILESIZE
2000
ubuntu@ip-172-31-4-185:~$ echo $HISTSIZE
1000
ubuntu@ip-172-31-4-185:~$
```

```
[ec2-user@ip-172-31-5-221 ~]$ echo $HISTFILE
/home/ec2-user/.bash_history
[ec2-user@ip-172-31-5-221 ~]$ echo $HISTFILESIZE
1000
[ec2-user@ip-172-31-5-221 ~]$ echo $HISTSIZE
1000
[ec2-user@ip-172-31-5-221 ~]$
```

To find, and use, previous commands, there are specific keys to perform various tasks:

Key	Usage
Up/Down arrow key	Browse through the list of commands previously executed
!!	Execute the previous command
CTRL-R	Search previously used commands

If you want to recall a command in the history list, but do not want to press the arrow key repeatedly, you can press CTRL-R to do a reverse intelligent search.

As you start typing the search goes back in reverse order to the first command that matches the letters you've typed. By typing more successive letters you make the match more and more specific.

The following is an example of how you can use the CTRL-R command to search through the command history:

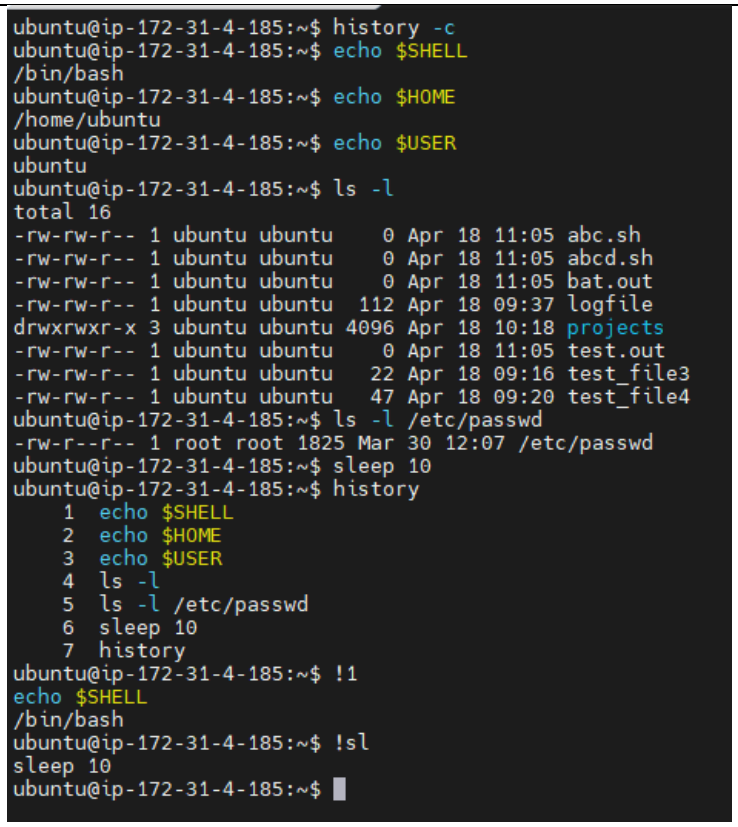
```
$ ^R                                    # This all happens on 1 line
(reverse-i-search)'al': alias            # Searched for 'al'; matched "alias"
$ alias                                 # Pressed Enter to execute the searched command
```

The table describes the syntax used to execute previously used commands.

Syntax	Task
!	Start a history substitution
!$	Refer to the last argument in a line
!n	Refer to the nth command
!string	Refer to the most recent command starting with string

History substitutions start with !. In the line `$ ls -l /bin /etc /var`, **!$** refers to `/var`, the last argument in the line.

To successfully perform the following test, make sure you first run **history -c** to clear out your history

$ history -c
$ echo $SHELL
$ echo $HOME
$ echo $USER
$ ls -l
 	
$ ls -l /etc/ passwd
 	
$ sleep 10
$ history
 	
$!1 # Execute command #1 above
echo $SHELL
/bin/bash
$!sl # Execute the command
# beginning with "sl" 	
sleep 10
	

Regular Expressions and Search Patterns

Regular expressions are text strings used for matching a specific pattern, or to search for a specific location, such as the start or end of a line or a word. Regular expressions can contain both normal characters or special characters, such as * and $.

Please note that this is just an introduction to regular expressions. There are whole books based on the subject.

These regular expressions are different from the wildcards (* and $) used in filename matching. The table lists search patterns and their usage.

Search Patterns	Usage
. (dot)	Match any single character
a | z	Match a or z
$	Match end of string
*	Match preceding item 0 or more times

Let's consider the following sentence:

the hairy black dog jumped over the little fence

We can apply the following search patterns to the sentence above:

Command	Usage
d..	matches **dog**
h.|l.	matches both **ha** and **li**
..$	matches **ce**
l.*	matches **little fence**
h.*y	matches **hairy**
the.*	matches the whole sentence

Search Contents of Files using grep

grep is extensively used as a primary text searching tool. It scans files for specified patterns and can be used with regular expressions as well as simple strings as shown in the table.

Command	Usage
grep [pattern] <filename>	Search for a pattern in a file and print all matching lines
grep -v [pattern] <filename>	Print all lines that do not match the pattern
grep [0-9] <filename>	Print the lines that contain the numbers 0 through 9
grep -n [^A-Z] <filename>	Print the lines that start with a capital letter preceded by their line numbers
grep -C 3 [pattern] <filename>	Print context of lines (specified number of lines above and below the pattern) for matching the pattern. Here the number of lines is specified as 3.

Before performing file content searches using grep, I will create a few more test files.

```
$ echo "This is test file #2." > test_file2
$ echo "This is test file #3." > test_file3
```

We will perform a few searches using the grep command. First, find the pattern **This** in files in the current directory.

```
$ grep This *
```

```
ubuntu@ip-172-31-4-185:~$ grep This *
grep: projects: Is a directory
test_file1:This is test file #1.
test_file1:This is the second line.
test_file2:This is test file #2.
test_file3:This is test file #3.
test_file4:This is test file #3
ubuntu@ip-172-31-4-185:~$
```

We limit the output by specifying a pattern to match and limiting the search to files whose names begin with **test**.

```
$ grep "file #[23]" test*
```

```
ubuntu@ip-172-31-4-185:~$ grep "file #[23]" test*
test_file2:This is test file #2.
test_file3:This is test file #3.
test_file4:This is test file #3
ubuntu@ip-172-31-4-185:~$
```

Now, we can prevent those matching lines from being displayed, by using the **-v** option

```
$ grep -v "test #[23]" test*
```

```
ubuntu@ip-172-31-4-185:~$ grep -v "file #[23]" test*
test_file1:This is test file #1.
test_file1:This is the second line.
ubuntu@ip-172-31-4-185:~$
```

Next, we will search for the pattern **alias** in **.bashrc** and include line numbers in the output.

```
$ grep -n alias .bashrc
```

```
ubuntu@ip-172-31-4-185:~$ grep -n alias .bashrc
75:# enable color support of ls and also add handy aliases
78:    alias ls='ls --color=auto'
79:    #alias dir='dir --color=auto'
80:    #alias vdir='vdir --color=auto'
82:    alias grep='grep --color=auto'
83:    alias fgrep='fgrep --color=auto'
84:    alias egrep='egrep --color=auto'
90:# some more ls aliases
91:alias ll='ls -alF'
92:alias la='ls -A'
93:alias l='ls -CF'
95:# Add an "alert" alias for long running commands.  Use like so:
97:alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo error)"
\s*//;s/[;&|]\s*alert$//''\'')"'
101:# ~/.bash_aliases, instead of adding them here directly.
104:if [ -f ~/.bash_aliases ]; then
105:    . ~/.bash_aliases
ubuntu@ip-172-31-4-185:~$
```

The next search is to find the port assigned to the SSHD service and include line numbers (**-n**) in the output, while ignoring case of the search using the **-i** option.

```
$ grep -i -n port /etc/ssh/sshd_config
```

```
ubuntu@ip-172-31-4-185:~$ grep -i -n port /etc/ssh/sshd_config
15:#Port 22
90:#GatewayPorts no
ubuntu@ip-172-31-4-185:~$
```

Our next search will be to view the SSHD configuration file settings without blank lines or lines beginning with number signs (#).

```
$ grep "^[^# ]" /etc/ssh/sshd_config
```

```
ubuntu@ip-172-31-4-185:~$ grep "^[^# ]" /etc/ssh/sshd_config
Include /etc/ssh/sshd_config.d/*.conf
PasswordAuthentication no
ChallengeResponseAuthentication no
UsePAM yes
X11Forwarding yes
PrintMotd no
AcceptEnv LANG LC_*
Subsystem       sftp    /usr/lib/openssh/sftp-server
ubuntu@ip-172-31-4-185:~$
```

The grep command is a great tool and has many options. For details, use grep's --help option or man page.

File Manipulation Utilities

In managing your files you may need to perform many tasks, such as sorting data and copying data from one location to another. Linux provides several file manipulation utilities that you can use while working with text files.

Let's first create a few test files that we will use in this section.

`echo "Janice Frank" > names1`
`echo "Bill Ford" >> names1`
`echo "Jill Jackson" >> names1`
`echo "Frank Jacks" >> names1`
`echo "Janice Frank" >> names1`
	`echo "Jack Sprat" > names2`
	`echo "Bill Ford" >> names2`
	`echo "Jill Jackson" >> names2`
	`echo "Frank Smithers" >> names2`
	`echo "Frank Jacks" >> names2`

`echo "E001 416-444-6655" > phone`
`echo "E002 416-888-9929" >> phone`
`echo "E003 514-228-7267" >> phone`
`echo "E004 514-323-7989" >> phone`
`echo "E005 416-333-7709" >> phone`
`echo "E006 514-727-0988" >> phone`
	`echo "416-444-6655 Toronto" > location`
	`echo "416-888-9929 Toronto" >> location`
	`echo "514-228-7267 Montreal" >> location`
	`echo "514-323-7989 Montreal" >> location`
	`echo "416-333-7709 Toronto" >> location`
	`echo "514-727-0988 Montreal" >> location`

The sort command is used to rearrange the lines of a text file either in ascending or descending order, according to a sort key. You can also sort by particular fields of a file. By default, sorts are in ascending order.

Syntax	Usage
sort <filename>	Sort the lines in the specified file
cat file1 file2 | sort	Append the two files, then sort the lines and display the output on the terminal
sort -r <filename>	Sort the lines in reverse order
sort -k <field #> <filename>	Sort the lines by key (field #) specified

When used with the -u option, sort checks for unique values after sorting the records (lines).

$ sort names1	ubuntu@ip-172-31-4-185:~$ sort names1 Bill Ford Frank Jacks Janice Frank Janice Frank Jill Jackson
$ sort names2	ubuntu@ip-172-31-4-185:~$ sort names2 Bill Ford Frank Jacks Frank Smithers Jack Sprat Jill Jackson
$ sort -k 2 -u names1	ubuntu@ip-172-31-4-185:~$ sort -k 2 -u names1 Bill Ford Janice Frank Frank Jacks Jill Jackson
$ sort -k 2 -u names2	ubuntu@ip-172-31-4-185:~$ sort -k 2 -u names2 Bill Ford Frank Jacks Jill Jackson Frank Smithers Jack Sprat ubuntu@ip-172-31-4-185:~$

You will notice from the results that, since the files contain a first and last name, we had to specify the 2nd field as the sort key.

uniq is used to remove duplicate lines in a text file and is useful for simplifying text display. uniq requires that the duplicate entries to be removed are consecutive. Therefore one often runs **sort** first and then pipes the output into **uniq**; if sort is passed the **-u** option it can do all this in one step.

To remove duplicate entries from some files, use the following command:

$ sort -k 2 names1 names2 | uniq > names

OR

$ sort -k 2 -u names1 names2 > names

```
ubuntu@ip-172-31-4-185:~$ sort -k 2 names1 names2 | uniq
Bill Ford
Janice Frank
Frank Jacks
Jill Jackson
Frank Smithers
Jack Sprat
ubuntu@ip-172-31-4-185:~$ sort -k 2 -u names1 names2
Bill Ford
Janice Frank
Frank Jacks
Jill Jackson
Frank Smithers
Jack Sprat
ubuntu@ip-172-31-4-185:~$ sort -k 2 names1 names2 | uniq > names
ubuntu@ip-172-31-4-185:~$
```

Now we have a file with employee names (**names**), as well as, a file with employee ids and phone numbers (**phone**). We will use the **paste** command to create a new file that contains all the data: name, employee ID and phone number.

paste can be used to create a single file containing all three columns. The different columns are identified based on delimiters (spacing used to separate two fields). The **-d** option allows you to specify a single delimiter or a list of delimiters to be used instead of tabs for separating consecutive values on a single line. Each delimiter is used in turn; when the list has been exhausted, paste begins again at the first delimiter.

First, we will use the default TAB as delimiter

$ paste names phone

```
ubuntu@ip-172-31-4-185:~$ paste names phone
Bill Ford       E001 416-444-6655
Janice Frank    E002 416-888-9929
Frank Jacks     E003 514-228-7267
Jill Jackson    E004 514-323-7989
Frank Smithers  E005 416-333-7709
Jack Sprat      E006 514-727-0988
ubuntu@ip-172-31-4-185:~$
```

Next, use space as delimiter

$ paste -d" " names phone

```
ubuntu@ip-172-31-4-185:~$ paste -d" " names phone
Bill Ford E001 416-444-6655
Janice Frank E002 416-888-9929
Frank Jacks E003 514-228-7267
Jill Jackson E004 514-323-7989
Frank Smithers E005 416-333-7709
Jack Sprat E006 514-727-0988
```

Now, use a comma as delimiter

$ paste -d, names phone

```
ubuntu@ip-172-31-4-185:~$ paste -d, names phone
Bill Ford,E001 416-444-6655
Janice Frank,E002 416-888-9929
Frank Jacks,E003 514-228-7267
Jill Jackson,E004 514-323-7989
Frank Smithers,E005 416-333-7709
Jack Sprat,E006 514-727-0988
```

Finally, revert back to the default delimiter of TAB and create a new file named phonebook

$ paste names phone > phonebook

$ cat phonebook

```
ubuntu@ip-172-31-4-185:~$ paste names phone > phonebook
ubuntu@ip-172-31-4-185:~$
ubuntu@ip-172-31-4-185:~$ cat phonebook
Bill Ford       E001 416-444-6655
Janice Frank    E002 416-888-9929
Frank Jacks     E003 514-228-7267
Jill Jackson    E004 514-323-7989
Frank Smithers  E005 416-333-7709
Jack Sprat      E006 514-727-0988
ubuntu@ip-172-31-4-185:~$
```

Now we have two files that contain employee phone numbers (**phonebook** & **location**). We can use the **join** command to combine the contents of both files into one file.

$ cat phonebook	$ cat location

`ubuntu@ip-172-31-4-185:~$ cat phonebook`
`Bill Ford  E001 416-444-6655`
`Janice Frank  E002 416-888-9929`
`Frank Jacks  E003 514-228-7267`
`Jill Jackson  E004 514-323-7989`
`Frank Smithers  E005 416-333-7709`
`Jack Sprat  E006 514-727-0988`
`ubuntu@ip-172-31-4-185:~$`
	`ubuntu@ip-172-31-4-185:~$ cat location`
	`416-444-6655 Toronto`
	`416-888-9929 Toronto`
	`514-228-7267 Montreal`
	`514-323-7989 Montreal`
	`416-333-7709 Toronto`
	`514-727-0988 Montreal`
	`ubuntu@ip-172-31-4-185:~$`

Since our files' common field is not in the same position, we must specify the location of each file's common field. The **phonebook** file has the common field at position 4, while the **location** file has the common field at position 1.

```
$ join -1 4 -2 1 phonebook location
```

```
ubuntu@ip-172-31-4-185:~$ join -1 4 -2 1 phonebook location
416-444-6655 Bill Ford E001 Toronto
416-888-9929 Janice Frank E002 Toronto
514-228-7267 Frank Jacks E003 Montreal
514-323-7989 Jill Jackson E004 Montreal
416-333-7709 Frank Smithers E005 Toronto
514-727-0988 Jack Sprat E006 Montreal
ubuntu@ip-172-31-4-185:~$
```

We can also format the output by using the **-o** option, along with the order of the file # and field # separated by commas.

```
$ join -1 4 -2 1 phonebook location -o 2.1,1.1,1.2,1.3,2.2
```

```
ubuntu@ip-172-31-4-185:~$ join -1 4 -2 1 phonebook location -o 2.1,1.1,1.2,1.3,2.2
416-444-6655 Bill Ford E001 Toronto
416-888-9929 Janice Frank E002 Toronto
514-228-7267 Frank Jacks E003 Montreal
514-323-7989 Jill Jackson E004 Montreal
416-333-7709 Frank Smithers E005 Toronto
514-727-0988 Jack Sprat E006 Montreal
ubuntu@ip-172-31-4-185:~$
```

```
$ join -1 4 -2 1 phonebook location -o 2.1,1.1,1.2,1.3,2.2 > emp_data
$ cat emp_data
```

```
ubuntu@ip-172-31-4-185:~$ join -1 4 -2 1 phonebook location -o 2.1,1.1,1.2,1.3,2.2 > emp_data
ubuntu@ip-172-31-4-185:~$ cat emp_data
416-444-6655 Bill Ford E001 Toronto
416-888-9929 Janice Frank E002 Toronto
514-228-7267 Frank Jacks E003 Montreal
514-323-7989 Jill Jackson E004 Montreal
416-333-7709 Frank Smithers E005 Toronto
514-727-0988 Jack Sprat E006 Montreal
ubuntu@ip-172-31-4-185:~$
```

split is used to break up (or split) a file into equal-sized segments for easier viewing and manipulation, and is generally used only on relatively large files. By default split breaks up a file into 1,000-line segments. The original file remains unchanged, and a set of new files with the same name plus an added prefix is created. By default, the x prefix is added. To split a file into segments, use the command `split infile`.

To split a file into segments using a different prefix, use the command `split infile <Prefix>`.

For testing purposes, we will create a large file, named big_file, using two existing log files. We use the **wc** program to report on the number of lines in the file. Then, we use **split** on **big_file** which contains over 4000 lines. First, let's do a directory listing to locate two large files:

```
$ ls -ltr /var/log/*.log
```

```
ubuntu@ip-172-31-4-185:~$ ls -ltr /var/log/*.log
-rw-r--r-- 1 syslog adm  257131 Apr 13 11:12 /var/log/cloud-init.log
-rw-r----- 1 root   adm    7373 Apr 13 11:12 /var/log/cloud-init-output.log
-rw-r--r-- 1 root   root   8450 Apr 14 06:09 /var/log/alternatives.log
-rw-r----- 1 syslog adm       0 Apr 17 00:00 /var/log/kern.log
-rw-r--r-- 1 root   root 102838 Apr 19 06:27 /var/log/dpkg.log
-rw------- 1 root   root   2844 Apr 19 07:00 /var/log/ubuntu-advantage-timer.log
-rw-r----- 1 syslog adm  357141 Apr 19 08:37 /var/log/auth.log
ubuntu@ip-172-31-4-185:~$
```

You see that **auth.log** & **cloud-init.log** are the largest files. We will use those two files to create **big_file**.

```
$ cat  /var/log/cloud-init.log  /var/log/auth.log > big_file
$ cat big_file | wc -l
```

```
ubuntu@ip-172-31-4-185:~$ cat /var/log/cloud-init.log /var/log/auth.log > big_file
ubuntu@ip-172-31-4-185:~$ cat big_file | wc -l
4413
ubuntu@ip-172-31-4-185:~$
```

After combining the two files into **big_file**, we see that it contains 4413 lines. Now, we will split the file which should result in four 1000 line segments and a segment with 413 lines. The segment names will begin with **part**.

```
$ split big_file part
$ ls -l part*
```

```
ubuntu@ip-172-31-4-185:~$ split big_file part
ubuntu@ip-172-31-4-185:~$ ls -l part*
-rw-rw-r-- 1 ubuntu ubuntu 146253 Apr 19 09:17 partaa
-rw-rw-r-- 1 ubuntu ubuntu 149726 Apr 19 09:17 partab
-rw-rw-r-- 1 ubuntu ubuntu 138732 Apr 19 09:17 partac
-rw-rw-r-- 1 ubuntu ubuntu 128337 Apr 19 09:17 partad
-rw-rw-r-- 1 ubuntu ubuntu  51944 Apr 19 09:17 partae
```

```
$ cat partaa | wc -l
```

```
ubuntu@ip-172-31-4-185:~$ cat partaa | wc -l
1000
```

```
$ cat partab | wc -l
```

```
ubuntu@ip-172-31-4-185:~$ cat partab | wc -l
1000
```

```
$ cat partac | wc -l
```

```
ubuntu@ip-172-31-4-185:~$ cat partac | wc -l
1000
```

```
$ cat partad | wc -l
```

```
ubuntu@ip-172-31-4-185:~$ cat partad | wc -l
1000
```

```
$ cat partae | wc -l
```

```
ubuntu@ip-172-31-4-185:~$ cat partae | wc -l
413
ubuntu@ip-172-31-4-185:~$
```

Miscellaneous Text Utilities

In this section, you will learn about some additional text utilities that you can use for performing various actions on your Linux files, such as changing the case of letters or determining the count of words, lines, and characters in a file.

The **tr** utility is used to translate specified characters into other characters or to delete them. The general syntax is as follows:

```
$ tr [options] set1 [set2]
```

The items in the square brackets are optional. **tr** requires at least one argument and accepts a maximum of two. The first, **set1** in the example, lists the characters in the text to be replaced or removed. The second, **set2**, lists the characters that are to be substituted for the characters listed in the first argument (**set1**). Sometimes these sets need to be surrounded by apostrophes (or single-quotes (')) in order to have the shell ignore that they mean something special to the shell. It is usually safe (and may be required) to use the single-quotes around each of the sets as you will see in the examples below.

For example, we will create a file named cities containing several lines of text in mixed case.

```
$ echo "Montreal Toronto Vancouver" > cities
$ cat cities
```

```
ubuntu@ip-172-31-4-185:~$ echo "Montreal Toronto Vancouver" > cities
ubuntu@ip-172-31-4-185:~$ cat cities
Montreal Toronto Vancouver
ubuntu@ip-172-31-4-185:~$
```

To translate all lower case characters to upper case, at the command prompt type:

```
$ tr [a-z] [A-Z] < cities
```

```
ubuntu@ip-172-31-4-185:~$ tr [a-z] [A-Z] < cities
MONTREAL TORONTO VANCOUVER
ubuntu@ip-172-31-4-185:~$
```

Our next task will be to translate spaces to tabs in the following string:

```
$ echo "This is for testing" | tr ' ' '\t'
```

```
ubuntu@ip-172-31-4-185:~$ echo "This is for testing" | tr ' ' '\t'
This    is      for     testing
ubuntu@ip-172-31-4-185:~$
```

We can also delete specified characters using -d option:

```
$ echo "this is another test" | tr -d 't'
```

```
ubuntu@ip-172-31-4-185:~$ echo "this is another test" | tr -d 't'
his is anoher es
ubuntu@ip-172-31-4-185:~$
```

Now, we will split a single line of words, separated by spaces, into new lines:

```
$ cat cities
$ tr -s ' ' '\n' < cities
```

```
ubuntu@ip-172-31-4-185:~$ cat cities
Montreal Toronto Vancounter
ubuntu@ip-172-31-4-185:~$ tr ' ' '\n' < cities
Montreal
Toronto
Vancounter
ubuntu@ip-172-31-4-185:~$
```

Delete any characters, but leave numbers (-c option replaces all characters **not** in SET1 (numbers))

```
$ echo "username: 123456" | tr -cd [0-9]
```

```
ubuntu@ip-172-31-4-185:~$ echo "username: 123456" | tr -cd [0-9]
123456ubuntu@ip-172-31-4-185:~$
```

The **tee** command takes the output from any command, and while sending it to standard output, it also saves it to a file. Note, appending to a file is done using the **-a** option. The **tee** command is very useful for testing scripts. Output is displayed onscreen, as well as, written to a file.

```
$ echo "Test #1" | tee testing
$ cat testing
$ echo "Test #2" | tee  -a testing
$ cat testing
```

```
ubuntu@ip-172-31-4-185:~$ echo "Test #1" | tee testing
Test #1
ubuntu@ip-172-31-4-185:~$ cat testing
Test #1
ubuntu@ip-172-31-4-185:~$ echo "Test #2" | tee -a testing
Test #2
ubuntu@ip-172-31-4-185:~$ cat testing
Test #1
Test #2
ubuntu@ip-172-31-4-185:~$
```

The **wc** (word count) command counts the number of lines, words, and characters in a file or list of files.

Option	Description
-l	display the number of lines
-w	display the number of words
-m	display the number of characters

```
$ cat testing

$ cat testing | wc -l
$ cat testing | wc -w

# NOTE: the newline character '\n' is
        on each line

$ cat testing | wc -m
```

```
ubuntu@ip-172-31-4-185:~$ cat testing
Test #1
Test #2
ubuntu@ip-172-31-4-185:~$ cat testing | wc -l
2
ubuntu@ip-172-31-4-185:~$ cat testing | wc -w
4
ubuntu@ip-172-31-4-185:~$ cat testing | wc -m
16
ubuntu@ip-172-31-4-185:~$
```

The **cut** command is used for manipulating column-based files and is used to extract (cut) specific columns. The default column separator is the tab character. A different delimiter can be given using command option **-d**.

```
$ cat emp_data
```

```
ubuntu@ip-172-31-4-185:~$ cat emp_data
416-444-6655 Bill Ford E001 Toronto
416-888-9929 Janice Frank E002 Toronto
514-228-7267 Frank Jacks E003 Montreal
514-323-7989 Jill Jackson E004 Montreal
416-333-7709 Frank Smithers E005 Toronto
514-727-0988 Jack Sprat E006 Montreal
ubuntu@ip-172-31-4-185:~$
```

```
# cut the first column
$ cat emp_data | cut -d" " -f1
```

```
ubuntu@ip-172-31-4-185:~$ cat emp_data | cut -d" " -f1
416-444-6655
416-888-9929
514-228-7267
514-323-7989
416-333-7709
514-727-0988
ubuntu@ip-172-31-4-185:~$
```

```
# cut the 1st, 2nd and 3rd columns
$ cat emp_data | cut -d" " -f1,2,3
```

```
ubuntu@ip-172-31-4-185:~$ cat emp_data | cut -d" " -f1,2,3
416-444-6655 Bill Ford
416-888-9929 Janice Frank
514-228-7267 Frank Jacks
514-323-7989 Jill Jackson
416-333-7709 Frank Smithers
514-727-0988 Jack Sprat
ubuntu@ip-172-31-4-185:~$
```

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

We discussed wildcard operators, regular expressions and search patterns. As you saw, these can be used to search for files on the system (using find), as well as, search the contents of files (using grep). We also worked with a number of utilities (sort, uniq, paste, join, split) to perform various actions on files. Finally, we used certain text utilities (tr, tee, wc, cut) that allowed us to control command output.

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