



ABC IT Education

WE'LL TAKE YOU FROM ZERO TO HERO IN A SNAP

Linux Systems Administration

Linux Intermediate 2

**Transferring & Copying Files over the
Network, Customizing the Shell Prompt &
Shell Aliases, Environment Variables,
Processes and Job Control**

Transferring & Copying Files over the Network

- From previous lessons, you have seen how to copy files from one location to another on the same system using the 'cp' command.
- In order to copy files from your local workstation to a Linux server or between Linux servers, you'll need to use either the 'scp' or 'sftp' command.
- SCP is secure copy and SFTP is secure file transfer protocol, also referred to as SSH file transfer protocol and both SCP and SFTP are extensions of the secure shell protocol.
- In order to use SCP or SFTP, especially in Windows, you need a client.
- Mac and Linux systems come with scp and sftp command line utilities not available in Windows.

- When running Windows, you can use the PuTTY Secure Copy Client - pscp.exe and the PuTTY Secure File Transfer client - psftp.exe.
- There are graphical clients as well.
- Cyberduck is popular for Mac and Windows,
- FileZilla runs on Mac, Windows and Linux.
- WinSCP is a Windows only SCP and SFTP client.
- If you are looking for a more interactive experience where you can examine the local files and remote files, use SFTP.
- With SCP, you need to know what files you want to transfer before issuing the command.

- SCP and SFTP are not the only ways to transfer files to remote systems, FTP, called file transfer protocol, can be used if enabled.
- To use FTP, you can use the built-in ftp command line client on Linux and Mac and a graphical client like WinSCP for windows.
- Note that FTP is not a secure transfer protocol like SCP and SFTP, meaning that your login credentials are sent in plain text over the network, and the files that you upload and download are not encrypted either.
- Always pick SCP or SFTP for your file transfer rather than FTP which is unsecure.

- Let's use our SFTP client (`psftp.exe`) to connect to this Linux server.
- When you connect to a remote server you're placed into your home directory, to verify type `'ls'` to see the files in the home directory.
- Prepending an `'l'` for local in front of the commands will work on the local directory, so `'lpwd'` will indicate that it is in the local/temp/test directory, while `'lls'` will show what files are on the local box in temp/test.
- Let's put `'ssh.txt'` onto the server. `psftp> put ssh.txt`
- Now do an `'ls'` and we can see that `'ssh.txt'` shows up on the remote server.

- Let's delete it off the remote server.
- And let's do an 'ls -l' for a long listing and see if it's gone, and it is gone.
- With SCP, you can copy from your local system to the remote system but you need to know the source and destination, so our source file will be 'ssh.txt'.
- The destination will be the server name or IP address, followed by a colon (:) at the end of the server name, then the path name. Uses local user name if one not specified.
- We could put this file in the /tmp directory on the remote server, or we could put this file in our home directory.
- Now connect through the Linux server with putty and make sure the file ssh.txt was placed in /tmp.

- Let's transfer a file to the remote server as a different user, and let us use 'adminuser@linuxsvr'.
- `scp ssh.txt adminuser@linuxsvr:/home/adminuser`
- And we'll place the file in the admin user's home directory.
- We can use Tilde as a shortcut for home or we can even specify the full path.
- It's the same with SFTP, let's connect to 'sftp adminuser@linuxsvr'.
- And then you can see that 'ssh.txt' was transferred.
- Let's look at using a graphical client like WinSCP.
- Select sftp protocol, enter host name or IP address, then user name and password, then login.

- If you're prompted about an SSH key, go ahead and accept it and this is a one time thing.
- On the left side, you see our local files on our workstation and on the right side are the remote files on the Linux server.
- We'll just pick a file to transfer up to our Linux server.
- Let's put it in our desktop folder.
- You can drag and drop the file from local to remote, also from remote to local.
- And you can also do a transfer by double clicking the file as well.

- Let's login to our Linux server and see if those files got transferred like we expected.
- And we can see that the file is on the desktop, double click to open.
- Let's look at our documents folder and we can see the file is there as well.
- In summary, to transfer files over the network, you need a client.
- There are command line clients like SCP, SFTP and FTP.
- You can also use a graphical client such as Cyberduck, FileZilla or WinSCP.
- If you have a choice to use FTP or SFTP, choose SFTP as it is more secure than FTP

Sftp commands

- ! run a local command
- bye finish your SFTP session
- cd change your remote working directory
- chmod change file permissions and modes
- close finish your SFTP session but do not quit PSFTP
- del delete files on the remote server
- dir list remote files
- exit finish your SFTP session
- get download a file from the server to your local machine
- help give help
- lcd change local working directory

Sftp commands

- `lpwd` print local working directory
- `ls` list remote files
- `mget` download multiple files at once
- `mkdir` create directories on the remote server
- `mput` upload multiple files at once
- `mv` move or rename file(s) on the remote server
- `open` connect to a host
- `put` upload a file from your local machine to the server
- `pwd` print your remote working directory
- `quit` finish your SFTP session
- `reget` continue downloading files

sftp commands

- **ren** move or rename file(s) on the remote server
- **reput** continue uploading files
- **rm** delete files on the remote server
- **rmdir** remove directories on the remote server

scp commands

- To copy a file from local to remote using scp

```
$ scp <local file> <remote user>@<remote server>:<remote dest>
```

<remote dest> can be a directory or a file. If it is a directory the local file name is maintained and if it is a file the <remote dest> file name is used.
- To copy a file from remote to local using scp

```
$ scp <remote user>@<remote server>:<remote file> <local dest>
```

<remote file> is a file and <local dest> is a directory or file. If it is a directory the remote file name is maintained and if it is a file then the <local dest> file name is used

scp commands

- To copy a directory from local to remote using scp

```
$ scp -r <local dir> <remote user>@<remote server>:<remote dir>
```

<local dir> is a directory and <remote dest> is a directory.
- To copy a directory from remote to local using scp

```
$ scp -r <remote user>@<remote server>:<remote dir> <local dir>
```

<remote dir> is a directory and <local dir> is a directory.

scp Options

- -r Recursively copy entire directories. Note that this follows symbolic links encountered in the tree traversal.
- -C Compression enable. Passes the -C flag to ssh to enable compression.
- -l limit – Limits the used bandwidth, specified in Kbit/s.

scp Options

- -o ssh_option – Can be used to pass options to ssh in the format used in ssh_config.
- -P port – Specifies the port to connect to on the remote host. Note that this option is written with a capital 'P'.
- -p Preserves modification times, access times, and modes from the original file.
- -q Quiet mode: disables the progress meter as well as warning and diagnostic messages from ssh.
- -v Verbose mode. Print debugging messages about progress. This is helpful in debugging connection, authentication, and configuration problems.

Customizing the Shell Prompt & Shell Aliases

- Default prompts can vary from system to system, and no matter what shell you're using, you can customize your prompt by simply setting an environment variable.
- For shells like Bash, ksh, and sh, the environment variable 'PS1' is used to set the primary prompt string.
- For shells like Csh, tcsh, and zsh, a 'PROMPT' environment variable is used.
- The format string you place in environment variable determines the look and feel of your prompt.
- Also, each shell uses slightly different format string, so consult the documentation for the shell you are using.
- Let's look at customizing the bash prompt since bash is the most popular default shell for user accounts on Linux systems.

- These are some of the most commonly used formatting string options for bash.
- `'\d'` = date in the weekday/month/date format.
- `'\h'` = hostname up to the first dot or the short hostname.
- `'\H'` = fully qualified domain name of the server.
- `'\n'` = new line, so you can have a multi line prompt.
- `'\t'` = current time in 24 hour format
- `'\T'` = current time in 12 hour format.
- `'\@'` = current time in 12 hour format with AM/PM included.
- `'\A'` = current time in 24 four hour format just using hours and minutes.

- `'\u'` = username of the current user.
- `'\w'` = current working directory.
- `'\W'` = give me the base name of the current working directory instead of the full path.
- `'\$'` = will let you know if you're normal user, it will display a dollar sign, or if you're the superuser or root account, it will display a pound sign.
- For a complete listing of all the formatting options, see the bash man page.
- To make your customized shell prompt persist between logins, add the PS1 value to your personal initialization files.
- Personal initialization files are commonly referred to as 'dot files' since they begin with a dot or period.

Persist PS1 Changes

```
$ echo 'export PS1="[\u@\h \w]\$ "' >> ~/.bash_profile
```

- Let's look at our current prompt string.
 - \$ echo \$PS1
- Now let's change it to 'username@host'.
 - \$ PS1="\u@\h \\$\" note this will only last for this session
- Let's add the time and the working directory to our prompt.
 - \$ PS1="<\t \u@\h \w> \\$\"
- Let's go to '/usr/bin' and you can see the current directory "/usr/bin" in the prompt.

- Let us now use 'W' for base name directories this time. Include the date, time and host.
 - `$ PS1="\d \t \h \W> \ $"`
- Now though we're in /usr/bin, it only says bin.
- Go back to your home directory, and remember that If you want to the prompt to persist between logins, we need to put these in our personalization files, usually .bash_profile.
- Set this environment variable in .bash_profile to show that it's going to work between logins. The prompt should have the username, hosts we're on, the directory we're in, and a '\$'.
- Exit your terminal and open anew one, you can see that the prompt is maintained between logins.

- How do you create multi line prompts? Let's make one.
- We will put date and time on one line, then the user host and working directory on the next line.
 - `$ PS1="< \d \t >\n[\u@\h \w] > \$"`
- Let's see it in action. Clear the screen here and you can see that every time that we hit enter, we get a new prompt with the date and current time on line one and a normal looking prompt on the second line.

Summary

- Shell prompts can be customized by setting the PS1 environment variable, and this differs from shell to shell.
- Bash, ksh, and sh, the environment variable used is 'PS1' and for other shells, Csh, tcsh, and zsh, a 'PROMPT' environment variable is used.
- To make your custom shell prompt to remain between logins, simply add the environment variable setting to your personal initialization files, '.bash_profile' preferred.

Shell Aliases

- What we are going to look at now is about creating, deleting and persisting aliases.
- There are some commands to use over and over again, so rather than typing the same command all the time, you can create a shortcut called, an alias.
- What is an alias? An alias can be thought of as a text expander, so creating aliases for commands that are really long or that you use often is a common practice.
- If you type 'ls -l' frequently, you may want to abbreviate it to 'll', and as a matter of fact, this alias often comes predefined on many Linux distributions.
- The 'alias' command can list or create aliases, so if you run alias without any additional information on the command line, it displays the list of aliases that are currently set.

- To create an alias, use 'alias' space, the alias name, equals (=) and the value for that alias.
- Aliases don't have to be used for just shortcuts, they are also usually used to fix common typing errors.
- If every time you try to type 'grep' and you end up typing 'grpe' you can create an alias and whenever you typed 'grpe' accidentally, alias will fix that for you and spell it 'grep'.
- In addition, you can also use aliases to make your work environment similar to that of another platform. For instance in windows, 'cls' clears a screen, but in Linux, the equivalent command is 'clear', so you can create an alias for clear
 - `$ alias cls="clear"`

- To remove an alias, type 'unalias' and the name of the alias.
 - `$ unalias`
- To remove all aliases, use 'unalias -a'.
- If you create aliases and log out and back again, the aliases you created would be lost.
- To make your aliases persist between sessions, add them to a dot files such as '.bashrc'.
- The downside to creating many aliases is that when you go from one system to another system that does not have your aliases, you may feel lost.
- If you want to work effectively on any Linux system that you have access to, keep your alias usage to a minimum.

- Another way to handle the situation is to simply copy your configuration files to each Linux system that you're going to work on.
- Let's say you run 'ls -l' a lot, then make this an alias, making it shorter.
 - `$ alias ll=ls -l`
- So when you type 'll' now, we'll get the same result as 'ls -l'.
- Remember, to list your aliases, just use the alias command, and to make them persist, you can edit your 'bash_profile' file and enter the aliases there.
- Before we finish this, remember that the alias command lists and creates aliases while the unalias command deletes aliases. To persist your aliases between logins, add them to your dot files.

Environment Variables

- What are environment variables and what are they used for? We will find out in this lesson and also find out how to create then and delete them.
- Environment variables are storage locations each with a name and a value and often affect how programs behave.
- In an earlier lesson we saw how Linux can tell where to search for commands by setting the `$PATH` environment variable.
- The easiest way to view all the environment variable names and values is to run the `'printenv'` command without any arguments.
- Running `'printenv'` you will see that each line contains the environment variable name, followed by an equals sign, and then the value of that environment variable.

- There are different kinds of the environment variables (i) informational environmental variable, like HOSTNAME, (ii) environmental variables whose value affect the way a program behaves, like PATH.
- In this case of the PATH variable, the shell uses the PATH to determine what paths to look for commands to execute and in what order, and if you change the PATH environment variable you change how the shell operates.
- Sometimes, you may want to look at the value of a specific environment variable. To do this, pass its name to the 'printenv' command as an argument .
- Example, you were to run "printenv USER", then only the value of the USER environment variable will be displayed on the screen.

- In Linux, most things are case sensitive, so the environment variables are also case sensitive.
- If you were to execute the command **\$ printenv User** in mixed letters, nothing will be returned because there is no environment variable named User in mixed letters.
- By convention, environment variables are usually in uppercase, though you can create lowercase environment variables that you will rarely find anywhere else.
- Apart from the 'printenv' command, another way to see the contents of an environment variable is to use the echo command.
- When using the echo command, you need to prepend a dollar sign (\$) to the environment variable name.

- To see the contents of the HOME environment variable using echo, you run "echo \$HOME".
- Never forget that environment variables are case sensitive thus \$ echo \$User is not the same as \$ echo \$USER so will not return a result.
- We create an environment variable with the "export" command followed by the environment variable name, an equals sign, and finally the value, in quotes, you want to assign to the variable.
- **DO NOT** use spaces around the equals sign as it will result in an error.
- Run the command 'export APPHOME="/home/user/app"' to create the APPHOME environment variable and assign to it the value "/home/user/app".

- Some commands look at the EDITOR environment variable, for example, to know which editor to start.
- If you set an environment variable that already exists, say the EDITOR environment variable, this would be overwritten, so this is also how you can update an environment variable.
- Another similar example is setting the TZ variable.
- If you run `export TZ="US/Pacific"`, this sets the Time Zone to the US/Pacific time, therefore the date command, for example, will return a different result depending on the value of the TZ variable.
- When you no longer need or want an environment variable, you remove it by running the `'unset'` command followed by the environment variable name.

- If for example we want to rely on the default time zone of the system instead of overriding that with the TZ environment variable, we remove it by running "unset TZ".
- Setting and unsetting environment variables from the command line effects your current session only.
- If you want to make your settings persist between sessions, you'll want to define your environment variables in your personal initializations files, like the .bash_profile file, using the same syntax as you learned.
- To explicitly set your timezone, you can add an export TZ line in your .bash_profile as shown below.
 - export TZ="US/Eastern"
- To see what environment variables are configured, we run the "printenv" command.

- Remember to pipe the output of your 'printenv' command to more or less (a pager utility), if you have too many environment variables to fit on one screen.
- With this, you now know what environment variables are configured in your environment and if you want to look at the value of just one of the variables, use printenv or echo.
- Take a look at the PATH with printenv PATH or echo \$PATH
- Now let's look at the current time with the date command, then set our time zone by changing the TZ environment variable to US/Pacific and check the date again.
- Now remove the TZ environment variable and see if our time zone is set back to the default.

- Now when we run the date command we're back in the US/Eastern time zone because that's the default set on this particular server.
- To display the time in the "US/Pacific" time zone and make this setting persist, add `export TZ="US/Pacific"` to your `~/.bashrc`.
- How to do it?
 - `$ vi .bashrc`
 - `export TZ="US/Pacific"`
 - Save the change and exit.
- If you log out and log back in, that environment variable will definitely be set, check it and sure enough, it's set to US/Pacific.

- Most Linux commands use environment variables and to tell what environment variables a program or command uses, look at command's man page in the ENVIRONMENT section.
- Type the command `$ man date` and look through the man page here for the ENVIRONMENT section you will see it has one environment variable listed, TZ.
- And it says that it specifies the time zone, unless overridden by command line parameters, so this is how you can tell what impact an environment variable will have on a given application.

- Let's Summarize
- You learned that environment variables are name/value pairs that can affect the behavior of an application.
- In addition, you learned how to view your environment with the `printenv` and `echo` commands.
- You also learned how to create or update environment variables using the `export` syntax, and talked about removing variables from your environment with the `unset` command.
- Then you learned how to make environment variables and their values persist between sessions.
- Lastly you learnt how how you can find what environment variables a given program uses by looking at it's man page.

Processes & Job Control

- In Linux like with other operating systems in use, processes are running all the time, so we will learn how to list the running processes or running programs and commands.
- We'll talk about the difference between foreground and background processes and how to manipulate those.
- Then we will talk about how to kill processes, display the currently running processes, using the 'ps' command.
- If you run the 'ps' command with no options, then ps will display the processes associated with your current session.

Listing Processes and Information

ps Display process status.

ps Options

-e	Everything, all processes.
-f	Full format listing.
-u username	Display username's processes.
-p pid	Display information for PID.

- To see every process including ones that are not owned or running as you, use ps with the '-e' option.
- Running 'ps' with the '-f' option gives you more information and a full format listing.
- To see processes running as a specific user, use the command 'ps -u username'.
- To display more information about a specific process, use its 'pid' (process identification number) as follows 'ps -p pid'.

Common ps Commands

<code>ps -e</code>	Display all processes.
<code>ps -ef</code>	Display all processes, full.
<code>ps -eH</code>	Display a process tree.
<code>ps -e --forest</code>	Display a process tree.
<code>ps -u username</code>	Display user's processes.

- Here are some common ps commands that you could use.
- 'ps -e' - display all the processes.
- 'ps -ef' - gives a full listing of all the processes.
- To display processes in a tree like format, use 'ps -eH' or '-e' with the '--forest' option.
- To display a specific users' process, use 'ps -u username'.

- Other ways to view processes and commands running on a system, you can use the 'pstree' command which displays processes in a tree format.
- You can use 'top' which is an interactive process viewer or you can use 'htop'. 'htop' is less common than top, so it may or may not be included in your Linux distribution.

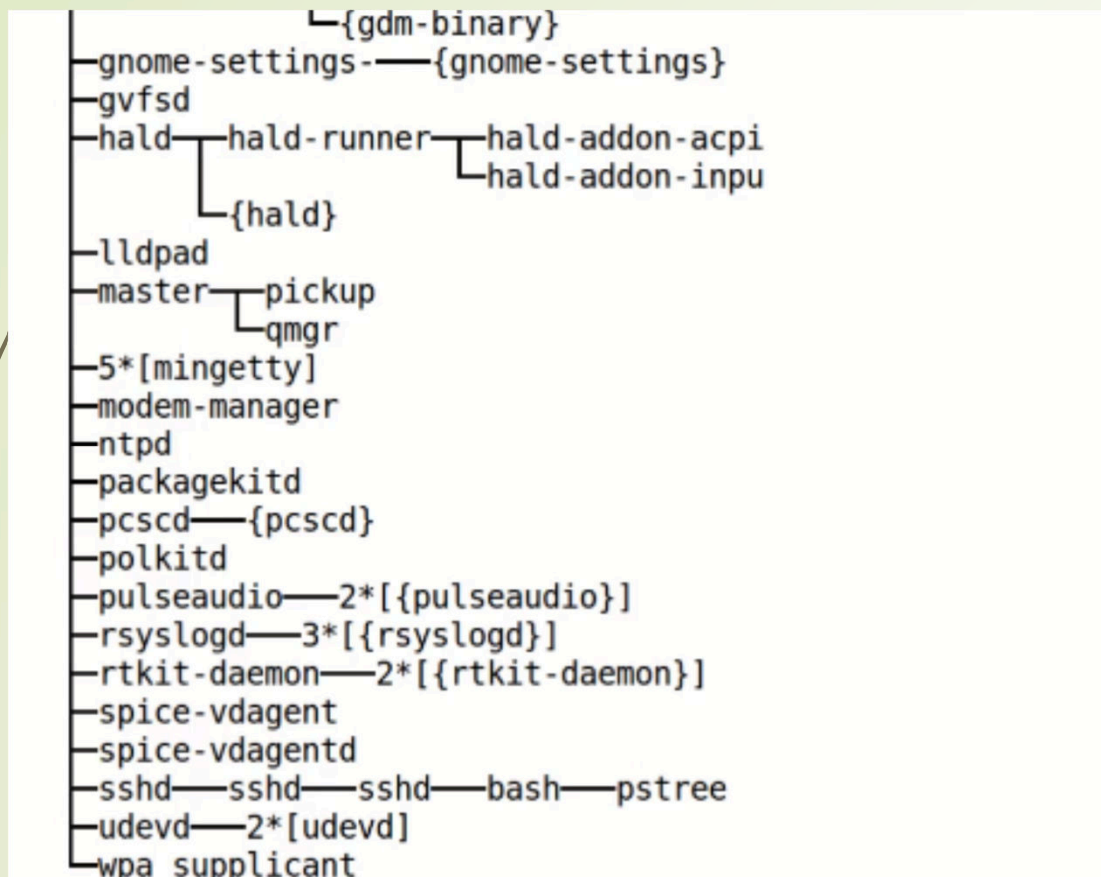
Other ways to view processes

<code>ps tree</code>	Display processes in a tree format.
<code>top</code>	Interactive process viewer.
<code>htop</code>	Interactive process viewer.

- To see a list of processes that are associated with your session, type `ps`.
- To look at a specific process, we can use '`-p`' and supply the 'pid' or process ID.

- And that will only list that one specific process.
- To get a full listing, we can use a '-f' option as it will tell you more information.
- You can do a '-e' for all the processes or '-ef' for a full listing and you can pipe it to less or more to see the output a page at a time.
- The information displayed is the user or UID that the process is running as, the PID, the process ID. The PPID, which is the parent process ID.
- Also, you have some more information about what time it started and commanding.
- To get a full process listing of a specific user, you can use this command with the user's user name as follows
 - `ps -fu <user name>`

- Let's look at the output in tree format.
 - `$ ps --forest`
- That's another way to do that. `$ ps -ef --forest`
- Here's an example of the 'pstree' command.



- Let's look at the top command.

```
Tasks: 119 total,  1 running, 118 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1750	jason	20	0	2696	1096	860	R	0.2	0.1	0:00.01	top
1	root	20	0	2900	1408	1180	S	0.0	0.1	0:00.81	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
4	root	20	0	0	0	0	S	0.0	0.0	0:00.01	ksoftirqd/0
5	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
6	root	RT	0	0	0	0	S	0.0	0.0	0:00.01	watchdog/0
7	root	20	0	0	0	0	S	0.0	0.0	0:00.10	events/0
8	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cgroup
9	root	20	0	0	0	0	S	0.0	0.0	0:00.00	khelper
10	root	20	0	0	0	0	S	0.0	0.0	0:00.00	netns
11	root	20	0	0	0	0	S	0.0	0.0	0:00.00	async/mgr
12	root	20	0	0	0	0	S	0.0	0.0	0:00.00	pm
13	root	20	0	0	0	0	S	0.0	0.0	0:00.00	sync_supers
14	root	20	0	0	0	0	S	0.0	0.0	0:00.00	bdi-default
15	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kintegrityd/0
16	root	20	0	0	0	0	S	0.0	0.0	0:00.11	kblockd/0

- It replaces the processes that are using the most CPU resources at the top of the list.
- Type a question mark for help and 'q' to exit.

- Let's look at the program called, 'htop'. It gives the same information as top, but just in a slightly different format.

CPU

|||||

Mem

|||||

Swp

12.5%

108/1006MB

0/2399MB

Tasks: 60, 81 thr; 1 running

Load average: 0.10 0.03 0.01

Uptime: 00:15:45

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
1	root	20	0	2900	1408	1180	S	0.0	0.1	0:00.81	/sbin/init
366	root	16	-4	2800	1064	344	S	0.0	0.1	0:00.26	/sbin/udev -d
848	root	16	-4	12932	760	560	S	0.0	0.1	0:00.00	auditd
847	root	16	-4	12932	760	560	S	0.0	0.1	0:00.00	auditd
864	root	20	0	35976	1484	992	S	0.0	0.1	0:00.00	/sbin/rsyslogd -i /var/run/
866	root	20	0	35976	1484	992	S	0.0	0.1	0:00.00	/sbin/rsyslogd -i /var/run/
867	root	20	0	35976	1484	992	S	0.0	0.1	0:00.00	/sbin/rsyslogd -i /var/run/
863	root	20	0	35976	1484	992	S	0.0	0.1	0:00.02	/sbin/rsyslogd -i /var/run/
909	root	20	0	2980	700	560	S	0.0	0.1	0:00.12	lldpad -d
934	root	20	0	2200	384	308	S	0.0	0.0	0:00.02	/usr/sbin/fcoemon --syslog
949	dbus	20	0	13532	1420	864	S	0.0	0.1	0:00.00	dbus-daemon --system
946	dbus	20	0	13532	1420	864	S	0.0	0.1	0:00.14	dbus-daemon --system
1008	root	20	0	20844	4196	3604	S	0.0	0.4	0:00.00	NetworkManager --pid-file=/
957	root	20	0	20844	4196	3604	S	0.0	0.4	0:00.02	NetworkManager --pid-file=/
962	root	20	0	4796	2296	1932	S	0.0	0.2	0:00.00	/usr/sbin/modem-manager
993	root	20	0	2020	576	492	S	0.0	0.1	0:00.00	/usr/sbin/acpid

Background and Foreground Processes

- Since we started our commands have been executing have been running in the foreground, and when a command, process or program is running in the foreground, the shell prompt will not be displayed until that process exits.
- For long running programs it can be convenient to send them to the background and backgrounded processes still execute and perform their task however they do not block you from entering further commands at the shell prompt.

Background and Foreground Processes

<code>command &</code>	Start command in background.
<code>Ctrl-c</code>	Kill the foreground process.
<code>Ctrl-z</code>	Suspend the foreground process.

Background and Foreground Processes

<code>bg [%num]</code>	Background a suspended process.
<code>fg [%num]</code>	Foreground a background process.
<code>kill</code>	Kill a process by job number or PID.
<code>jobs [%num]</code>	List jobs.

- To start a process in the background, place an ampersand (&) at the end of the command.
- To kill the current foreground process, hold down the control key and press the letter 'c', and to suspend the process, type 'ctrl z' and that will stop the process.

- A process that is suspended is not running in the background, it is actually stopped.
- To send a suspended process to the background, use the 'bg' command.
- You can background a specific job number by preceding it with a percent sign (%) for example, '%3'.
- If you don't supply a job number, bg will operate on the current job.
- To list your jobs type the jobs command.

Killing Processes

<code>Ctrl-c</code>	Kills the foreground proc.
<code>kill [-sig] pid</code>	Send a signal to a process.
<code>kill -l</code>	Display a list of signals.
<code>kill 123</code>	
<code>kill -15 123</code>	
<code>kill -TERM 123</code>	<code>kill -9 123</code>

- There are two ways to kill a running process. If the process is running in the foreground, simply type 'ctrl c' to kill it.
- The other way to kill a process is with the kill command.
- You run kill by just supplying the 'pid' or the process ID.

- You can also specify a signal to send to the process, and the default signal used by kill is termination.
- You'll see the signal referred to as 'sig' term or term for short.
- Signals have numbers that correspond to their names, and the default term signal is number 15, so running kill 123 is the same as running 'kill -15 123' or 'kill -term 123'.
- If a process does not terminate when you send the term signal, use the kill signal which is number 9.
- Start a program in the background, by ending the command line with an ampersand (&).
- When a command is run in the background, 2 numbers are displayed, the job number, in brackets, which can be referenced by proceeding it with a percent sign (%).

- The second number is the pid, so we run 'ps -p', which will give us the pid, say pid of 3328, we'll see the process that was started in the background.
- If we type jobs, you'll see that 1 is the job number for number 1, you can also reference it by %1.
- We run the foreground command to bring this job to the foreground.
- And to kill a foreground process we can type 'ctrl c' and that will stop the process.
- Now start multiple jobs in the background, then type the jobs command here.
- The plus sign (+) in the jobs output represents the current job while the minus sign (-) represents the previous job.

- The current job is considered to be the last job that was stopped while it was in the foreground or the last job started in the background.
- If you run the fg and bg commands and no information is supplied then the current job, which can be explicitly referred to as `'%%'` or `'%+'`, is operated upon.
- The previous job can be referred to by `'%-'`.
- For instance we can run jobs to list all your jobs.
- `$ jobs %%` - will list the current job
- `$ jobs %+` - will also list the current job
- `$ jobs %-` = will list the previous job
- To bring a job back to the foreground, type fg and a job number so `$ fg %2` will bring job 2 to the foreground.

- Now let us kill job 2 sing Ctrl+c.
- The list the jobs to see if job 2 is gone and yes, now we have jobs number 1, 3, and 4, so job 2 was killed.
- Bring job number 1 to the foreground \$ fg %1, we can suspend it by typing 'Ctrl+z'.
- Running the jobs command shows job 1 as stopped. It's not in the background, it is actually stopped and not dling any work.
- To send job 1 to the background, type bg and then '%1' and running the command 'jobs' shows jobs 1, 3 and 4 running in the background.
- Let's kill the job 1 by job number, \$ kill %1, you'll notice that job 1 is terminated. .

- Let's foreground job number 3, we can do that with the fg command `$ fg %3`, but you can also foreground a job by simply typing the job number, like `$ %3`.
- **The kill command**
- To list all the signals use the command `$ kill -l`. This are the signals that can be sent to processes.
- Kill defaults to signal 15 which is sigterm.
- To kill a process, use 'kill -9' or 'kill -KILL' on that process.
- Start couple of jobs and note their PIDs (process IDs)
- Let's kill one of the processes with 'kill <PID>', and the job gets terminated.

- Let's do a 'kill -9' for another PID, kill -9 <PID>
- And sure enough that killed that job as well.

➤ **Summary**

- To display information about running processes and programs, use the ps command.
- To see all the processes running on a system with a full listing, use 'ps -ef'.
- To look at a process for a given user, use 'ps -u' followed by the account name.
- To kill a process that's in the foreground, type 'ctrl c'.
- To suspend a foreground process, type 'ctrl z'.

- To background a process, type `bg`.
- And to foreground a process, use the `fg` command.
- To display information about your running jobs, use the `jobs` command.
- You can use the `kill` command to kill jobs by job number with a percent sign or you can kill programs and processes by `pid`.