1.1

Monday, February 27, 2017 9:59 AM

The vi editor is a utility that creates and modifies text files. It is the standard command line text editor included with most Linux distributions. The vi editor uses the following modes:

Command mode is the initial mode vi uses when started. It provides commands that can cut and replace text. It is also the mode from which you access the other vi modes.
Command line mode is used to load files and to save files after editing them in the file system.
Edit mode is the mode that vi uses to write and edit text in the file. It has two operation modes:
Insert mode adds text between the preceding and subsequent text.
Replace mode overwrites subsequent text.
The table below lists some of the most common vi commands:

Use	To	Mode
vi	Start vi. Type the command at the shell prompt.	N/A
vi [file_name]	Start vi and immediately begin working on the named file (either a new, non-existent file or an existing file). Type the vi command at the shell prompt.	N/A
Insert key		
i		
s	Enter insert mode from command mode.	Command
Esc key	Enter command mode from edit mode.	Insert/Replace
Delete key	Delete text.	Insert/Replace
Insert key	Toggle between insert and replace mode while in edit mode.	Insert/Replace
#[line_number]	Go to a specific line in the document while in command mode. For example #94, moves the cursor to line 94.	Command
dw	Cut a whole word and trailing space.	Command
de	Cut a whole word but omit the trailing space.	Command
d$ or D	Cut all text following the cursor to the end of the line.	Command
dd	Cut a line from the text.	Command
p	Place text in memory into the document.	Command
u	Undo the last action.	Command
O	Open a new line above the current line.	Command
o	Open a new line below the current line.	Command
Ctrl+g	Display the file name, the total number of lines in the file, and the cursor position.	Command
/[term]	Search forward for all instances of a term. Press n to go to the next term and N to go to the previous term.	Command
?[term]	Search backward for all instances of a term. Press n to go to the previous term and N to go to the next term.	Command
yy	Copy a line of text into memory.	Command
a	Append text after the cursor.	Command
A	Append text after the current line.	Command
C	Change text from current cursor position to the end of the line.	Command
cc	Change text of the entire line.	Command
ZZ	Save current file and exit vi.	Command
h	Move the cursor one space to the left.	Command
j	Move the cursor down a line.	Command
k	Move the cursor up a line.	Command
l	Move the cursor one space to the right.	Command
z	Exit without saving.	Command

:	Enter command line mode from command mode.	Command
w	Save the current document.	Command line
w [file_name]	Name and save the file.	Command line
w![file_name]	Overwrite the file.	Command line
q	Exit vi. This produces an error if the text was modified.	Command line
q!	Exit vi without saving.	Command line
wq or exit	Save the document and exit vi.	Command line
e!	Reload the file from the last saved version. This discards all edits and reloads the last saved version of the file into vi.	Command line

1.2

Monday, April 3, 2017 10:23 AM

An alias is a shortcut stored in memory that runs a command on your Linux system. Most distributions include predefined aliases that are created at system startup. However, a custom alias can be defined from the shell prompt. Be aware of the following:

Aliases defined with the alias command are not persistent across reboots.
To make aliases persistent across reboots, add the alias definitions to /etc/profile or home/user/.bashrc.
The following table describes the commands that create and remove aliases:

Use	To	Example
alias	Display the currently defined aliases on the system.	[rtracy@fs5 ~]$ alias alias egrep='egrep --color=auto' alias fgrep='fgrep --color=auto' alias grep='grep --color=auto' alias l.='ls -d .* --color=auto' alias ll='ls -l --color=auto' alias ls='ls --color=auto' alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'
alias [name='command']	Create a custom alias that runs an existing command. A single alias can be defined to run multiple commands. When creating the alias, encapsulate the command(s) with quotation marks or apostrophes.	alias securebackup='cp ./*.* /dev/st0/*.*;shred -fuvz ./*' creates an alias that copies all files in the current directory to a tape backup device and then shreds the original files. alias forcelogout="killall /usr/bin/Xorg" creates a shortcut that kills all Xserver processes.
unalias [name]	Remove an alias.	unalias securebackup removes the alias specified for the securebackup command. unalias forcelogout deletes the forcelogout alias.

1.3

Monday, April 3, 2017 10:24 AM

An environment variable is a setting that defines the user's computing environment. Be aware of the following details:

By convention, you should use uppercase characters when defining and referencing variables names, called variable identifiers (e.g., SHELL and EUID).
Environment environmental variables are initially assigned default values from the shell configuration files.
You can modify the default values assigned to environment variables to customize the user's computing environment.
If you modify the default value assigned to an environment variable, that change will apply only to the current session. You must export the modified variable so change applies to all other shell sessions.
To make changes to environment variables persistent across system reboots, add the change to the appropriate shell configuration file(s).
You can define non-environment variables at the shell prompt that can be used to store data. These are called user-defined variables. User-defined variables are commonly defined when writing shell scripts.
The table below lists common environment variables:

Variable	Description
BASH	The location of the bash executable file.
SHELL	The user's login shell.
CPU	The type of CPU.
DISPLAY	Location where X Windows output goes.
ENV	The location of the configuration file for the current shell.
EUID	The ID number of the current user.
HISTFILE	The file name where past commands are stored.
HISTSIZE	The number of past commands that HISTFILE stores for the current session.
HISTFILESIZE	The number of past commands that HISTFILE stores for the multiple sessions.
HOME	The absolute path of the user's home directory.
HOST	The name of the computer.
HOSTNAME	HOSTNAME is identical to HOST, but is only used on certain distributions.
INFODIR	The path to the computer's information pages.
LOGNAME	The username of the current user.
MAIL	The path to the current user's mailbox file.
MANPATH	The path to the computer's man pages.
OLDPWD	The directory the user was in prior to switching to the current directory.
OSTYPE	The type of operating system. Usually, this is Linux.
PATH	The directory paths used to search for programs and files.

Use a colon to separate entries in the PATH variable.
Do not include a period (.) in the PATH variable. A period indicates that the working directory is in the path, and poses a security risk.

PS1	The characters the shell uses to define what the shell prompt looks like.
PWD	The path of the current working directory.
LANG	The language the operating system uses.

The table below lists the most common environment variable commands:

Use	To	Examples
echo $[variable]	View the variable's value.	echo $SHELL displays the user's default shell.
env	Display the values for environment variables applied to child sessions.	

set Set shell environment variables. Without options, set displays the set environment variables for the system.
unset [variable] Remove an environment variable. unset HOMEDIR removes the HOMEDIR variable.
[VARIABLE]=[value] Change the value assigned to a variable.
To append information to an environment variable instead of replacing it, include the current variable in the command. For example, PATH=$PATH:/bin/additionalpath.
HOMEDIR=/projects gives the HOMEDIR variable a value of /projects.
export [variable] Export a variable to make it available to all other shell sessions. export HOMEDIR makes the HOMEDIR variable available to all other shell sessions.

PATH=$PATH:/bin/special ; export PATH appends a directory to PATH and immediately exports the variable.

1.4

Monday, April 3, 2017 10:24 AM

Shell configuration files are scripts that execute when a shell starts. The shell type determines which shell configuration files are executed.

Login shells run when the system starts and is using only the CLI as the user interface.
Non-login shells run when a shell session is opened from within the GUI. Because the user authenticated when they initially logged into the graphical environment, they are not required to log in when accessing a shell session.
The following are the names of the files used when the shell starts:

Configuration File	Description	Used by shell type
~/.bashrc	~/.bashrc stores shell preferences for individual users. (This file is also typically called by login shell configuration files on most distributions.)	Non-login
/etc/profile	/etc/profile stores system-wide configuration commands and is used primarily to set environment variables.	Login
~/.bash_profile	~/.bash_profile stores shell preferences for individual users.	Login
~/.bash_login	~/.bash_login stores commands that execute when a user logs in.	Login
~/.profile	~/.profile stores configuration preferences similar to /etc/profile, but for individual users.	Login
~/.bash_logout	~/.bash_logout stores commands that execute when a user logs out.	Login

Login shells execute the configuration scripts they use in the following order:

/etc/profile
~/.bash_profile (If this file is found, the shell does not look for additional configuration script files.)
~/.bash_login (If this file is found, the shell does not look for additional configuration script files.)
~/.profile (This file executes only in the absence of the two preceding configuration script files.)
The su -l command switches to a user and creates a new login shell; however, without the -l option, a non-login shell is started.

As you study this section, answer the following questions:

What is the difference between redirection and piping?
When might you choose to redirect the input of a command?
What are the three default file descriptors that Linux uses to classify information for a command?
How can you overcome the 128 KB shell command size restriction?
After finishing this section, you should be able to complete the following tasks:

Redirect the standard output from the screen to a file.
Redirect and append new content to an existing file.
Redirect a standard error from a command to a file.
Redirect the standard input to a command.
Pipe the output of a command to the input of another command.
Use the pipe command to search a file for specified text.
Use the pipe command to create a text stream.
This section covers the following TestOut Linux Pro certification exam objectives:

3.0 File System Management

Create, copy, move, and delete directories in the file system.
Create, copy, move, and delete files in the file system.

Administrators often use the following methods to manipulate information created by shell commands:

MethodDescription
Redirection	Redirection directs standard input, output, and error streams to locations other than the default. Be aware of the following redirection details:
By default, the Linux system classifies information with the following file descriptors:
Standard input (stdin) comes from the keyboard. Using redirection, 0 represents stdin.
Standard output (stdout) displays on the monitor. Using redirection, 1 represents stdout.
Standard errors (stderr) display on the monitor. Using redirection, 2 represents stderr.
Linux commands use the greater than symbol (>) to show redirection of output, the less than symbol arrow (<) to indicate redirection of input, and the double greater than symbol (>>) to append output to an existing file.
If you use a single greater than symbol (>) to redirect output to a file and that file already exists, its contents will be overwritten. To preserve the contents of the file, use the double greater than symbol (>>).
The tee command writes to standard output and redirects it to a file at the same time.
Piping	Piping directs the output of one command into the input of another command. Pipes:
Use the pipe symbol (|).
Can combine several commands to make a text stream.
The following table shows the results of several redirection and piping commands:

Example	Result
ls /usr > /tmp/deletemeWrites the list of files in the /usr directory to a file named /tmp/deleteme.
ls /nonesuch > /tmp/deleteme	Does not write anything to a file and sends an error message to the monitor '/nonesuch not found' (because the directory does not exist).
ls /nonesuch 2 > /tmp/deleteme	Writes the standard error message to a file named /tmp/deleteme. This will only happen if an error message is actually generated by the command. If not, then the stderr will be null.
ls /bin /nonesuch > /tmp/deleteme	Writes the contents of the /bin directory to the /tmp/deleteme file, but sends the error message '/nonesuch not found' to the screen.
ls /bin /nonesuch > /tmp/deleteme 2>&1	Directs the standard output to the /tmp/deleteme file, then directs that the standard error messages be sent to the same place as the standard output. Both the list of files in the /bin directory and the error message are written to the same file.
ls /bin /nonesuch 2>&1 > /tmp/deleteme	Writes the contents of the /bin directory to the /tmp/deleteme file, but sends the error message '/nonesuch not found' to the screen. This is because standard error messages are directed to the same place that standard output goes, but this is before standard output has been directed to the file.
ls /bin >> /tmp/deleteme	Appends the list of files from the /usr directory on to the end of the /tmp/deleteme file.
sort < unordered_file.txt > ordered_file.txt	Takes input from the unordered_file.txt file sends it to the sort command, and then writes a new file named ordered_file.txt.
cat /usr/wordlist1 /usr/wordlist2 | sort	Sends the output of the cat command, the contents of wordlist1 and wordlist2, to the input of the sort command. The result is a sorted list of the combined contents of wordlist1 and wordlist2.
cat /usr/wordlist1 /usr/wordlist2 | mail jdoe	Mails the combined list of words in wordlist1 and wordlist2 to the user jdoe.
ls /bin | sort | mail jdoeLists the contents of /bin then sorts the combined contents and mails them to jdoe.
cat /usr/wordlist1 /usr/wordlist2 | sort | tee sortedwordlist	Lists the contents of wordlist1 and

wordlist2 then sorts the combined contents, then sends the results to the monitor and a file named sortedwordlist.
cat /usr/wordlist1 | tee log.txt   Writes to the standard output and the log.txt file.

The bash shell allows you to use command substitution. Using command substitution, you can run a shell command and have the output from that command sent back to the command line as an argument for another shell command. Command substitution first creates a child
process that runs the first shell command. The stdout from the first command is redirected back to the shell, which parses it into words separated by white space. After the first command has finished running, the shell starts another child process that runs the second command, using parsed output from the first command as arguments.

For example, to use the cat command to display all the files in /var that contain the text "ens32", you could enter cat $(fgrep -l ens32 /var/*). First, the fgrep -l command is run to search through all the files in the /var directory for the text string "ens32". Using the -l option with fgrep causes the command to return just a list
of matching file names. This list of files is then sent as arguments to the cat command, which then displays the contents of each file it received from fgrep.

The xargs command reads items from the standard input and breaks up long lists of arguments into smaller, usable lists. Xargs:

Makes it easier to pipe input into commands that take arguments.
Overcomes a 128 KB shell command size restriction imposed by the Linux kernel.
Is commonly used in conjunction with the following commands:
find
ls
locate
grep -l
The following table describes the common xargs options:

Use	To	Examples
-0	Ignore space names in files.	find / -print0 -name *.odt | xargs -0 rm deletes all .odt files in the file system, even those with spaces in the file names.
-I [variable]	Replace the initial argument of a command with the argument from the standard input.	find / -name '*.jpg' | xargs -I var1 cp var1 /home/user/Pictures finds all the .jpg files on the computer and copies them into the /home/user/Pictures directory.

1.5

Monday, April 3, 2017 10:24 AM

Monday, February 27, 2017
10:23 AM

Wi-Fi tethering - 13.62 mbps download, 2.56 mbps upload, with an average ping of 115 ms
USB tethering - 20 mbps download, 4.76 mbps upload, with an average ping of 95 ms
Bluetooth tethering - 1.6 mbps download, 0.65 mbps download, with an average ping of 152 ms

Home=/root
LANG= en_us.UTF-8
SHell= /bin/bash
TERM linux

1.6

Monday, April 3, 2017 10:24 AM

The following table describes several basic commands that can be used to manage files:
Command Description Examples
touch If the file does not exist, it creates a blank version of the file. If the file does exist, it updates the file's modification and last accessed times.
touch myfile makes a blank file named myfile.
cat Displays the contents of a file in the shell. This command can display multiple files at once.
cat myfile displays the contents of myfile.
cat myfile yourfile displays the contents of myfile and yourfile together.
less Displays the contents of a file, pausing one screen at a time.
Use the Spacebar to scroll to the next screen.
Use the Up arrow and Down arrow to scroll up and down.
Press q to exit.
less bigfile displays the contents of bigfile one screen at a time so it can be read.
head Lists the first 10 lines of a specified file. Use the -n option to specify a specific number of lines to display.
head /home/user/myfile lists the first 10 lines of myfile.
head -n 20 /home/user/myfile lists the first 20 lines of myfile.
head -n -35 /home/user/myfile displays all lines in myfile, omitting the last 35 lines.
tail Lists the last 10 lines of a specified file.
-n specifies a specific number of lines.
-f monitors the file.
tail /home/user/myfile lists the last 10 lines of myfile.
tail -n 20 /home/user/myfile lists the last 20 lines of myfile.
tail -n -15 /home/user/myfile displays all lines in myfile, omitting the first 15 lines.
tail -f /var/firewalld displays the last 10 lines of /var/firewalld and then dynamically displays new lines in the file as they are added.
file Shows the file type. The file command is useful because Linux does not require file extensions.
The file command uses file signatures in:
/usr/share/misc/magic
/usr/share/misc/magic.mgc
/etc/magic
file myfile shows whether myfile is a text, data, xml, or other type of file.
cp Copies files. Copying leaves the source file intact.
-f overwrites files that already exist in the destination directory.
-i prompts before overwriting a file in the destination directory.
cp /temp/document_ab.txt ~/doc/document.txt copies document_ab.txt from the /temp directory to the ~/doc directory and renames the file to document.txt.
cp /temp/*.txt ~/doc copies all text files from the /temp directory to the ~/doc directory.
mv Moves or renames files (and directories). Moving files erases the source file and moves it to the destination.
-f overwrites files that already exist in the destination directory.
-i prompts before overwriting a file in the destination directory.
-n never overwrites files in the destination directory.
mv /temp/document.txt ~/doc/document.txt moves document.txt from the /temp directory to the ~/doc directory.
mv /temp/*.txt ~/doc/*.txt copies all text files from the /temp directory to the ~/doc directory.
rm Removes a file or directory. Use the -f option to delete with a prompt.
The rm command deletes a file or directory's inode, but it does not actually delete its data. To

permanently remove data, use the shred command.
rm myfile deletes a file in the current directory named myfile.
rm /home/user/myfile deletes myfile from the /home/user directory regardless of the current directory.
rm -f /home/user/temp/* deletes all files in the temp directory with prompts.
shred Deletes the file and overwrites the file's data on the hard disk. The shred command is useful when deleting files that contain proprietary information or other sensitive data.
-n specifies the times to overwrite. The default is 25 times.
-u deletes the inode.
-v display the progress of the file deletion.
-z overwrites the filename with zeros.
shred -u -z companysecrets.txt deletes companysecrets.txt, overwrites the file with random information, then leaves zeros in place of the file.
lsattr Lists file attributes.
-R recursively list attributes of directories and their contents.
-V displays the program version.
-a lists all files in directories.
-d lists directories like other files, rather than listing their contents.
-v lists the file's version/generation number.
lsattr /etc/grub/grub.conf lists the attributes of the grub.conf file.

1.7 Redirection and Piping

Monday, April 3, 2017 10:24 AM

Administrators often use the following methods to manipulate information created by shell commands:

Method	Description
Redirection	*Redirection* directs standard input, output, and error streams to locations other than the default. Be aware of the following redirection details: • By default, the Linux system classifies information with the following file descriptors: • Standard input (stdin) comes from the keyboard. Using redirection, 0 represents stdin. • Standard output (stdout) displays on the monitor. Using redirection, 1 represents stdout. • Standard errors (stderr) display on the monitor. Using redirection, 2 represents stderr. • Linux commands use the greater than symbol (**>**) to show redirection of output, the less than symbol arrow (**<**) to indicate redirection of input, and the double greater than symbol (**>>**) to append output to an existing file. If you use a single greater than symbol (>) to redirect output to a file and that file already exists, its contents will be overwritten. To preserve the contents of the file, use the double greater than symbol (>>). • The **tee** command writes to standard output *and* redirects it to a file at the same time.
Piping	*Piping* directs the output of one command into the input of another command. Pipes: • Use the pipe symbol (**|**). • Can combine several commands to make a text stream.

The following table shows the results of several redirection and piping commands:

Example	Result
ls /usr > /tmp/deleteme	Writes the list of files in the **/usr** directory to a file named **/tmp/deleteme**.
ls /nonesuch > /tmp/deleteme	Does not write anything to a file and sends an error message to the monitor '/nonesuch not found' (because the directory does not exist).
ls /nonesuch 2 > /tmp/deleteme	Writes the standard error message to a file named **/tmp/deleteme**. This will only happen if an error message is actually generated by the command. If not, then the stderr will be null.
ls /bin /nonesuch > /tmp/deleteme	Writes the contents of the **/bin** directory to the **/tmp/deleteme** file, but sends the error message '/nonesuch not found' to the screen.
ls /bin /nonesuch > /tmp/deleteme 2>&1	Directs the standard output to the **/tmp/deleteme** file, then directs that the standard error messages be sent to the same place as the standard output. Both the list of files in the **/bin** directory and the error message are written to the same file.

ls /bin /nonesuch 2>&1 > /tmp/deleteme	Writes the contents of the /bin directory to the **/tmp/deleteme** file, but sends the error message '/nonesuch not found' to the screen. This is because standard error messages are directed to the same place that standard output goes, but this is before standard output has been directed to the file.
ls /bin >> /tmp/deleteme	Appends the list of files from the **/usr** directory on to the end of the **/tmp/deleteme**file.
sort < unordered_file.txt > ordered_file.txt	Takes input from the unordered_file.txt file sends it to the sort command, and then writes a new file named **ordered_file.txt**.
cat /usr/wordlist1 /usr/wordlist2 | sort	Sends the output of the cat command, the contents of wordlist1 and wordlist2, to the input of the sort command. The result is a sorted list of the combined contents of **wordlist1** and **wordlist2**.
cat /usr/wordlist1 /usr/wordlist2 | mail jdoe	Mails the combined list of words in **wordlist1** and **wordlist2** to the user jdoe.
ls /bin | sort | mail jdoe	Lists the contents of /bin then sorts the combined contents and mails them to jdoe.
cat /usr/wordlist1 /usr/wordlist2 | sort | tee sortedwordlist	Lists the contents of **wordlist1** and **wordlist2** then sorts the combined contents, then sends the results to the monitor and a file named **sortedwordlist**.
cat /usr/wordlist1 | tee log.txt	Writes to the standard output and the log.txt file.

The bash shell allows you to use command substitution. Using command substitution, you can run a shell command and have the output from that command sent back to the command line as an argument for another shell command. Command substitution first creates a child process that runs the first shell command. The stdout from the first command is redirected back to the shell, which parses it into words separated by white space. After the first command has finished running, the shell starts another child process that runs the second command, using parsed output from the first command as arguments.
For example, to use the cat command to display all the files in /var that contain the text "ens32", you could enter **cat $(fgrep -l ens32 /var/*)**. First, the fgrep -l command is run to search through all the files in the /var directory for the text string "ens32". Using the -l option with fgrep causes the command to return just a list of matching file names. This list of files is then sent as arguments to the cat command, which then displays the contents of each file it received from fgrep.
The **xargs** command reads items from the standard input and breaks up long lists of arguments into smaller, usable lists. Xargs:

- Makes it easier to pipe input into commands that take arguments.
- Overcomes a 128 KB shell command size restriction imposed by the Linux kernel.
- Is commonly used in conjunction with the following commands:
- **find**
- **ls**
- **locate**

- **grep -l**

  The following table describes the common **xargs** options:

Use	To	Examples
-0	Ignore space names in files.	**find / -print0 -name *.odt | xargs -0 rm** deletes all .odt files in the file system, even those with spaces in the file names.
-I [variable]	Replace the initial argument of a command with the argument from the standard input.	**find / -name '*.jpg' | xargs -I var1 cp var1 /home/user/Pictures** finds all the .jpg files on the computer and copies them into the **/home/user/Pictures** directory.

The following table describes basic commands you can use to manage directories on a Linux system:
Command Description Examples
pwd Displays the current working directory.
If a user named Fred is currently in his home directory and typed pwd at the shell prompt, /home/Fred is displayed.
cd Changes the present working directory.
cd .. changes to the parent directory.
cd / changes to the root directory.
cd directory1 changes to a directory named directory1 within the current working directory. (This is a relative path.)
cd /home/Fred/directory1 switches directory1 in the user Fred's home directory, regardless of the current working directory. (This is an absolute path.)
ls Displays the contents of a directory. Options include:
-a displays all directory contents, including hidden content.
-l displays extended information, including the owner, modified date, size, and permissions.
-R displays the contents of a directory and all of its subdirectories.
-d displays directories but not files.
-r reverses the sort order.
ls -al displays a long listing of all the contents in the current working directory, including hidden content.
ls -d only displays directories within the current directory.
ls -R /etc displays the contents of the /etc directory and all of its subdirectories.
mkdir Creates a new directory. Use the -p option to create all directories within the specified path that do not already exist.
mkdir work_files creates a directory named "work_files" in the current working directory.
mkdir /home/Fred/work_files creates a directory named "work_files" within the specified path.
cp -r
cp -R Copies directories. Copying leaves the source contents (directories and files) intact.
cp -r /temp /home/user copies the entire /temp directory (with all of its files, subdirectories, and files in the subdirectories) to the /home/user directory.
mv Moves or renames directories (and files). Moving directories erases the source directory and places it in the destination. Options include:
-f overwrites a directory that already exist in the destination directory.

-i prompts before overwriting a directory in the destination directory.
-n never overwrites files in the destination directory.
mv /temp/station ~/doc/ moves station from the /temp directory to the ~/doc directory.
rmdir   Deletes an empty directory.
rmdir ~/Fred/work_files deletes the work_files directory if it is empty.
rm   Removes the directory (and file) inode, but not actually delete the data. Options include:
-r deletes directories (and all files) in the directories.
-f deletes without prompting.
rm -rf /home/user/temp deletes the temp directory with all its subdirectories and files without prompting.
rm -r /home/user/* deletes all directories and files in the /home/user directory.

From <http://www.sildom.me/SS/2017-03-08_16-27-49.txt>

1.8 and 1.9

Monday, April 3, 2017 10:27 AM

Initrd

https://en.wikipedia.org/wiki/Runlevel
https://en.wikipedia.org/wiki/Initrd
https://ss64.com/bash/chmod.html

Windows has services
Linux has daemons

Pwd	MKDIR	Current_user	Alias	Xargs
Ls	Rmdir	Vi	Env	Grep
Cat	Pipes>less,more,head,tail	Touch	Echo	Sort
Cd	Redirection	Gedit	Set	Runlevel
Uname	Find	Su	Ln > Hardlink and softlink	systemctL
Cp	locate	Whoami		
Rm				
Mv				

1.10

Monday, April 3, 2017 10:28 AM

Links are files that point to another file. Linux uses two types of links that you need to be familiar with:

Type	Description
Hard link	A *hard link* is a duplicate entry in the file system that points to a specific piece of data on the disk drive. With a hard link: • Duplicate file inodes are used. The *inode* specifies where a file's data physically exists on a disk. With a hard link, the link file and the original file both share the same inode. • The data stored in the link file is exactly the same as the data in the original file. • The data is preserved within the link file even if the original file is deleted. • A hyphen (-) is used as the first character in the permission string, which is the same character used for normal files (e.g., **-**rwxr-xr-x).
Symbolic link	A *symbolic link* (also known as a *soft link*) is a file that points to another file in the file system. A symbolic link is similar to shortcuts in the Windows OS. With a symbolic link: • Separate inodes are used. The link file has an inode that is distinct from the inode of the file being pointed to. • A lower-case "L" (l) is used as the first character in the permission string (e.g., **l**rwxrwxrwx indicates a symbolic link).

The following commands are used to create hard links and symbolic links:

Command	Description	Examples
ln *[source] [link_name]*	Creates links. • **ln -s** creates a symbolic link to a file. • **ln -b** creates backup of a file. • **ln -i** determines the inode for hard or symbolic links. • **ln** (with no options) creates a hard link between files.	• **ln /home/jsmith/project1 /home/edunford/project1** creates a hard link to /home/jsmith/project1 in /home/edunford/. • **ln -s /home/jsmith/project1 /home/edunford/project1_ln** creates a symbolic link named /home/edunford/project1_ln that points to /home/jsmith/project1. • **ln -s /home/jsmith/project1 /home/edunford/project1_ln** creates a symbolic link named /home/edunford/project1_ln that points to /home/jsmith/project1. • **ln -i** displays the inodes for the contents in the present working directory. • **ln -b /home/jsmith/file1 /bup** copies file1 as file1~ in /bup.
cp *[source] [link_name]*	Copies files and creates links. • **cp -l** creates hard links rather than copying the files. • **cp -s** creates symbolic links rather than	• **cp -l /home/jed/fil1 /home/esam/proj1** creates an exact copy of /home/jed/fil1 in /home/esam/. • **cp -s /home/mkon/text /home/ytew/text_ln** creates a symbolic link named /home/ytew/text_ln that points to /home/mkon/text.

	copying the files.	

From <http://cdn.testout.com/client-v5-1-10-395/startlabsim.html?culture=en-us>

1.11

Monday, April 3, 2017 10:28 AM

The Filesystem Hierarchy Standard (FHS) defines a consistent file system for Linux systems by defining a standard set of directories, subdirectories, and files. FHS is a subset of the Linux Standards Base (LSB) which is an organization and a set of guidelines for promoting a set of standards to increase Linux distribution compatibility.

Directory	Description
/	The **/** character represents the root directory of the Linux system. All other directories are located beneath the **/** (root directory) of the system.
/bin	The **/bin** directory contains binary commands that are available to all users.
/boot	The **/boot** directory contains the kernel and bootloader files.
/dev	The **/dev** directory contains device files that represent the devices used by the system, such as a hard drive, mouse, and printer.
/etc	The **/etc** directory contains configuration files specific to the system.
/home	The **/home** directory contains (by default) the user home directories.
/lib	The **/lib** directory contains shared program libraries and kernel modules.
/media	The **/media** directory is used to mount removable media, such as optical discs and USB drives.
/mnt	The **/mnt** directory is used for temporarily mounting remote file systems.
/opt	The **/opt** directory contains additional programs on the system.
/proc	The **/proc** directory contains information about the system state and processes.
/root	The **/root** directory is the root user's home directory. Do not confuse **/root** with the root of the file system (**/**).
/sbin	The **/sbin** directory contains system binary commands.
/srv	The **/srv** directory contains files for services such as HTTP and FTP servers.
/sys	The **/sys** directory contains the *sysfs* virtual file system which displays information about devices and drivers.
/tmp	The **/tmp** directory contains temporary files created by programs during system use.
/usr	The **/usr** directory contains system commands and utilities.
/var	The **/var** directory contains data files that change constantly, such as: • User mailboxes • Print queues • Log files

From <http://cdn.testout.com/client-v5-1-10-395/startlabsim.html?culture=en-us>

1.12

Monday, April 3, 2017 10:28 AM

Use the following commands to find file locations:

Command	Description	Examples
find	Searches through all files based on the file system by name, file size, time created, and other options. Be aware of the following **find** command options: • **-name** locates a file or directory by name in a specific path. When using **-name**: • Enclose name strings in single quotes. • Use wildcards for partial names. • Use **-iname** to ignore case. • **-user** finds files owned by a specific user. • **-size** finds files of a specific size. Use the following options: • **c** for bytes • **k** for kilobytes • **M** for megabytes. • **-mtime** finds files last modified before or after a specified number of days ago. • **-type [*f* or *d*]** specifies whether to find files or directories. • **-maxdepth** specifies how many levels down to search. • -**print0** finds filenames with spaces. • **-o** specifies the *or* parameter when searching with multiple criteria. • **.** (period) specifies the search locations as the current directory and subdirectories.	• **find /user/home -name '*.txt'** finds all files with an extension of .txt in the /user/home directory. • **find / -name '*paper*'** looks through the entire directory for any filename or directory name that contains "paper" (e.g., termpaper.odt or wallpaper.jpg). • **find /user/home -size -300k** finds all files in the /user/home directory smaller than 300 K. • **find /user/home -size +300k** finds all files in the /user/home directory larger than 300 K. • **find /user/home -mtime -5** finds all files in the /user/home directory modified within the last five days. • **find / -type f -name '*paper*'** finds only filenames that contain "paper". • **find / -type d -name '*paper*'** finds only directory names that contain "paper". • **find -maxdepth 3 / -name '*.txt'** finds .txt files three directory levels down from the root directory. • **find -print0 -name '*.txt'** finds .txt files with spaces in the name (e.g., myreport.txt and my report.txt). Without the -**print0**option, only files without spaces are listed.
locate	Searches for files in the file system. The **locate** utility maintains a database containing a listing of all the files and directories in the file system. Be aware that the **locate**command: • Searches through an index instead of the actual file system. Because of this, the locate command usually runs much faster than the find command. • Searches all files if no path is specified. • Maintains an index of all files and directories in the **/var/log/locatedb** file. • Uses the **updatedb** command to update the index. (The **/etc/updatedb.conf** file is used to configure for **updatedb**.) The index is updated on a regular schedule. This means it is possible for locate to find a file in the index that no longer exists in the actual file system. Likewise, it is possible for locate not to find a new file in the file system that has not been added to the index yet. Be aware of the following **locate** command options: • **-c** counts the number of entries rather than list them. • **-e** lists files only after verifying that they exist. • **-i** ignores case.	• **updatedb** updates the index file (/var/log/locatedb). • **locate /user/home paper** locates all files with the string "paper" as any part of the filename or directory path under the /user/home directory. • **locate lib** locates all files with the string "lib" anywhere in the filename or directory path. • **locate -c lib** counts the number of files with the string "lib" (e.g., 46512). • **locate -e .odt** verifies that all .odt files listed in the file index actually exist before it lists them. • **locate -i LibraryFines.csv** finds the libraryfines.csv file regardless of case. • **locate -l 25 lib** lists only the first 25 files from the file index that contain the string "lib". • **locate -b lib** displays /var/lib and /user/home/libraryfines.csv but not /var/lib/usbutils/usb.ids. If the search pattern contains no globbing characters, such as the wildcard character (*), locate behaves as if the pattern begins and ends with a wildcard. For example, searching for "paper" is the same as searching for "*paper*"

	• **-l** limits the number of files listed. • **-b** searches for the string in only file or directory base names.	same as searching for "*paper*".
which	Displays the path to a command and determines whether a package is installed.	• **which ls** shows the path to the **ls** binary (executable) file. • **which photorec** shows the path to the **photorec** binary file if **photorec** is installed.
whereis	Displays the path to a Linux command's binary files, manual pages, and source code (if sources are installed). Be aware of the following whereis command options: • **-b** lists the path to the binary file (similar to **which**). • **-m** lists the path to the man page files. • **-s** lists the location of the source code. • **-u** lists entries that do not have source code, binary file, and man page locations. When no options are specified, **whereis** shows all available data.	• **whereis ls** lists information about the **ls** command.
type	Displays a command's type. Possible categories include: • A built-in shell command • A command that the shell calls • An aliased command • A function If a called command has been used recently, the output says that the command is *hashed*, which means that it is in the shell's hash table.	• **type cd** displays "cd is a shell built-in". • **type more** displays the path to the binary file for **more**.

The term *file globbing* refers to the use of wildcards (e.g., *****, ***.***, *.txt) to match specific files.

The **grep** command searches through file text for specific words or character patterns. The following table describes the**grep**, **egrep**, and **fgrep** commands and several of their options:

Command	Description	Examples
grep	Searches through files for a specified character string. By default, **grep** is context sensitive and displays the string in the context of the line containing the string. • **-A *[number]*** prints a specified number of lines following the matching lines. • **-a** searches binary (executable) files as though they were text files. • **-B *[number]*** prints a specified number of lines before the matching lines. • **-C *[number]*** prints a specified number of lines of context around the matching lines. • **-c** shows the number of matches of the string for the file. • **-E** uses regular expressions for the text pattern. • **-e *[pattern]*** specifies a literal pattern. • **-f** searches for multiple strings using a file that lists the string patterns. • **-F** uses a file as the source for the string patterns. • **-i** ignores the case of the string.	• **grep -A 3 Midway ~/docs/WWII-report** searches WWII-report for the pattern "Midway" and prints the line and the next three lines. • **grep -a var11 /bin** searches all files, including binary files, in the /bin directory for the pattern "var11". • **grep -c 3 Midway ~/docs/WWII-report** shows only the number of times the pattern "Midway" is found in the WWII-report file. • **grep -e '--count' ~/docs/doc1** looks for the pattern "--count" in the doc1 file rather than interpreting it as an option. • **grep -l -r Midway ~/docs** shows the name of all files in the /home/user/docs directory that contain the term "Midway". • **grep -m 2 battle ~/docs/WWII-report** shows only the first two times the term "battle" is found in the file. • **grep -n -i customVariable1**

	• **-l** lists just the names of the files with a match. This is used when search multiple files. • **-m *[number]*** shows only a specified number of matches for a file. • **-n** displays the line number of the lines containing the term. • **-r** searches the directory and all subdirectories for files containing the term. • **-v** displays non-matching lines. • **--include=*[file_name]*** searches only in files with names that match a specified string. • **--exclude=*[file_name]*** searches in files with names that do *not* match a specified string. • **-w** searches for whole words only.	**~/java/program1.java** shows the line numbers of lines that have the term "customVariable1" in the program1.java file. This ignores the case. • **grep -r battle ~/docs/** searches the directory and all subdirectories for the term "battle". • **grep -w tank ~/docs/WWII-report** searches only for the whole word "tank" in the file.
egrep	Uses regular expressions in the search strings. The **egrep** command uses the same options and syntax as **grep** and is identical to **grep -E**. Constructors for **egrep**regular expressions include: • **^** matches terms that occur at the beginning of a line. • **$** matches terms that occur at the end of a line. • **\<** matches words that begin with the term. • **\>** matches words that end with the term. • **[*asdf*]** matches any one of the characters in the brackets. • **[*0-9*]** matches any of the range of numbers 0-9. • **[^*xyz*]** omits any one of the letters in the list • **.** matches any single character. • **[*asdf*]+** matches one or more of the characters in the list. • ***** matches any number, or none of the preceding single character • **|** matches either of the terms. • **** displays the literal value of a character used for expressions. • **()** groups expressions.	• **egrep ^FAILURE ~/error_logs** matches the term "FAILURE" when it is at the beginning of the line in error_logs. • **egrep tty7$ ~/.bash_history** matches the term "tty7" when it is at the end of the line. • **egrep \<are ~/myfile** matches all words or strings that begin with "are" (e.g., "are", "area", and "arena"). • **egrep \>are ~/myfile** matches all words or strings that end with "are" (e.g., "are", "hare", and "aware"). • **egrep watche[ds] ~/myfile** matches either "watched" or "watches". • **egrep exhibit[0-9] ~/myfile** matches "exhibit1", "exhibit3", or "exhibit8". • **egrep [^Xx]mas ~/myfile** matches "Christmas" but not "xmas" or "Xmas". • **egrep .are ~/myfile** matches "hare" and "care", but not "aware" or "are". • **egrep file[0-9]+ ~/myfile** matches "file0", "file10", and "file15636". • **egrep fil* ~/myfile** matches "fil", "filll", and "filllllllllllllll". • **egrep fil.* ~/myfile** matches "file", "fill", "file102", and "filings". • **egrep men|women ~/myfile** matches "men" or "women". • **egrep Hello\? ~/myfile** matches "Hello?".
fgrep	Uses a file as the source for the string patterns. When searching for fixed strings, rather than regular expressions, **fgrep**: • Uses the same options as the **grep** command and has the same syntax. • Is identical to **grep -F**, but searches faster than **grep**. • Interprets the pattern as a list of fixed strings, any of which can be matched.	• **fgrep Midway Nimitz ~/docs/myfile** searches myfile for lines containing "Midway" or "Nimitz".

From <https://cdn.testout.com/client-v5-1-10-405/startlabsim.html?culture=en-us>

2.1.3

Monday, February 27, 2017 10:00 AM

Planning and designing a Linux installation has the following advantages:

- A plan ensures that the installer knows exactly what should happen during the installation. The plan places all the information in the installer's hands before installation begins.
- There are fewer variables involved when diagnosing and resolving problems.
- The plan gives the IT team a point of reference when changes are requested that will require the project scope, resources, or schedule to be modified.

The following table describes the general steps of an effective installation design:

Step	Description
Perform a needs assessment	An effective needs assessment determines the goals of the installation, creates a plan to meet those needs, and measures the results of the plan. This involves: • Interviewing managers to determine the goals they want to achieve, the problems they need to solve, and the results they expect. • Writing clear, measurable statements that specifically address the goals. • Identifying the stake holders. • Confirming correct authorization. • Aligning the installation with current organizational strategy and technology. • Verifying funding. • Creating a support strategy. • Determining the scope by: • Identifying deadlines. • Determining the tasks that must be completed. • Planning for human resource allocation.
Pick a distribution	Picking a distribution involves: • Determining whether the computer should be a server or a workstation. Most distributions can be either, but some are better designed for specific functions and even specific types of servers. • Determining whether end users will be comfortable with the distribution, or whether configuration changes might be necessary. • Determining whether required software is available for the distribution. Make a list of applications and ensure that they are provided on the distribution. • Ensuring that the distribution has the necessary support.
Determine the hardware requirement s	Some computer hardware is incompatible with some distributions. Ensure that: • The computers' hardware is on the distribution's hardware compatibility list. • The computers have sufficient CPU speed, memory, and other system requirements to run the distribution and the installed software. • The distribution's architecture matches the computer's CPU architecture. These include: • x86 for 32-bit CPUs • x64 for 64-bit CPUs • IA-64 for Itanium CPUs • ALPHA for Alpha CPUs

	• PPC for Power PC (Apple) CPUs
Plan the file system	The file system determines how a computer's files are organized on a hard drive. Linux supports several file system types that have different characteristics, including: • *ext2* has volume integrity features that may take several minutes to run after a system crash. • *ext3* uses journaling to ensure that only incomplete transactions are checked after a system crash. It is the default file system on most distributions. • *ReiserFS* also uses journaling and also implements additional security features based on its file structure. Pick the file system that best meets your organization's needs. Considerations include: • Maximum volume size • Maximum file size • File name size • Permissions and file security • Encryption support • Recovery support and speed • Backup support • Journal support
Plan the partitions	Partition planning is another element of file system planning. An efficient strategy is to create multiple partitions based on the types of files held on the partition and the user access needs to the data. Consider creating separate partitions for the following directories, and set mount options based on the type of files in the directory: • **/** (root) must to be at least 8 GB, but really should be much larger. The partition holding the root directory should not be on an extended MBR partition and must be formatted with a Linux file system. • **/home** should be as large as needed to store the user files. • **/boot** should be 100–200MB. It needs to be in the first 1024 cylinders of the disk for older BIOS versions. • **/opt** should be large enough to support the applications that will be installed. • **/tmp** should be large enough to store the temporary files created by the system. Be aware that these temporary files are periodically cleaned out by the operating system. • **/usr** should be large enough to support the packages to be installed on the system. • **/var** should be large enough to support the user mail boxes, print queues, and log files that will be created on the system. • **swap** should be two times larger than the amount of installed RAM on desktop systems and one times the amount of installed RAM on server systems. Linux can use either a swap file or a swap partition for the swap area. Whenever possible, use a separate swap partition. To increase performance, consider creating multiple swap partitions on multiple storage devices. **/etc**, **/bin**, **/sbin**, **/lib**, **/dev**, and **/proc** must be all on the same partition (/). These directories have system configuration files that are necessary for Linux to function properly.
Identify software	Determine which software packages need to be installed, and install only those packages. This ensures that system resources are conserved, and that

	vulnerabilities are limited.
Identify the users	Determine the users who will use the computer. Consider the following: • Ensure correct name spelling for the users. • Determine whether users log in locally or over the network. • Define a list of groups to which the users will belong. The root user is always installed. Use this account only when necessary to ensure security.
Gather network information	Gather the following types of network information: • Protocol (IPv4 or IPv6) • IP address • Default gateway • Subnet mask • Server information for DNS servers, mail servers, and other network servers • Network topology information, such as domain names • Naming conventions for servers and workstations • Domain names
Select an installation source	Installation sources include: • CD • DVD • ISO file • Network share • Online installation repository

From <https://cdn.testout.com/client-v5-1-10-407/startlabsim.html?culture=en-us>

The following table describes the general steps necessary to install a Linux operating system:

Step	**Description**
Start the install	In many cases, this step involves booting the computer from installation media. The BIOS may need to be configured to boot from the media. Linux installers offer several installation options: • New install • Reinstall • Upgrade • Installation recovery • Reduced feature installation
Select a language	The selected language becomes the default language for all users. This can be changed later.
Set the system time	The system time can be set to local time or Coordinated Universal Time (UTC) time. • For networks dispersed over multiple time zones, choosing UTC simplifies administration tasks. • Some distributions have options to synchronize the time over the network.
Format partitions	A *partition* is a logical division of a storage device associated with a hard disk drive. Create the partition structure according to the specifications in the installation plan. Determine the directory structure and file system type for each partition. This may require initializing a hard drive. Partitioning options include the following: • Use all space (makes a single partition from the entire disk drive)

	• Replace existing Linux system(s) (re-partition**s** all the previously installed Linux Operating systems) • Shrink partitions (reduces the size of an existing partition making room for additional partitions) • Use free space (creates a partition from unpartitioned space on the disk drive) • Create custom layout (manually creates partitions according to the specific needs of the system or administrator) Keep the following in mind when creating partitions: • Initializing a hard drive removes all existing data. • A *swap* partition is required for virtual memory. Swap partitions should be between one to two times larger than the computer's installed memory. • Linux computers can only have four partitions; however, a single extended partition can be sub-divided into additional partitions. • Create separate partitions for the following directories to keep logs or abnormally large user files from taking all disk space and to make recovery of data easier should the operating system crash. • /home (user directories) • /opt (installed software) • /var (log files) The operating system can be reinstalled on the root partition (/), and the others can then be remounted with no loss to data.
Select applications and services	Installing applications and services depends on the role of the system. Applications and services include the following: • The *boot loader* determines which operating system boots by default (if more than one operating system exists on a computer). • *Package patterns* include packages necessary for a specific computer role (e.g., Graphical Desktop or Web Server). • *Package repositories* are locations on the Internet where software packages are maintained. Specific Linux utilities search and install software automatically from these package repositories.
Set the root password	The secure password for the root user (and any other user) should have the following characteristics: • At least 8 characters long (longer passwords are harder to crack) • Use a combination of letters, numbers, and symbols • Should not include a username or a dictionary word (or common variations)
Specify a host name	The name of the computer identifies the computer on a network. A domain may be required.
Configure network connections	Configure the network connections.
Configure services	Some services must be configured based on the role of the system (e.g., a Web Server).
Add new users and groups	Create user accounts and groups for the users who will use the computer. • Installations usually require at least one standard user account. • Network login options enable the system to access a server for login information rather than maintaining local authentication information.

Configure the hardware settings	Hardware configuration settings might require appropriate drivers or language settings.
Identify remote access needs	Depending on your job role, you may need to manage Linux systems that don't have external peripherals connected, including monitors, keyboards, or mice. This a common configuration for both Linux servers and virtualized Linux systems that run on a hypervisor. To manage these types of systems, you must access them remotely from another device. There are several options available to accomplish this: • If the system has a management interface integrated within its hardware, you can use it to access the system display from a web browser and perform management tasks. This type of hardware is typically found only on high-end server systems. • If the system does not have an integrated management interface, you can use two network protocols to remotely access it. • Use SSH to securely access the shell prompt of the system and run commands as if you were sitting in front of the system. You can also tunnel X server traffic through the SSH connection, which allows you to securely access the graphical desktop of the system. • You can use Virtual Network Computing (VNC) to remotely access the graphical desktop of the system.

From <https://cdn.testout.com/client-v5-1-10-407/startlabsim.html?culture=en-us>

A *locale* is a set of files that Linux uses to determine country and language specific settings for various applications. Locales:

- Determine the way data displays on a computer.
- The language and encoding of the text displayed on screen
- Character classes
- Sort order
- Number formatting
- Currency type and format
- Date and time display
- Use configuration files that are part of the system library and are located in **/usr/share/locale** on most distributions.
- Use language codes specified in ISO-639 and country codes specified in ISO-3166.
- Use the following command format: *[Language[_[territory].[codset-modifier]* (e.g., **en_GB.UTF-8** and **de_DE.euro**). You can use codeset codes such as ASCII and UTF-8.

ISO-8859 was an early standard designed to extend the ASCII character set. It uses an eighth bit to extend the ASCII character set by 128 characters. This allows the character set to support extended characters used by some languages. The ISO-8859 standard is divided into several sub-standards to support specific languages. For example, ISO-8859-8 supports Hebrew, while ISO-8859-7 supports Greek.

Today, the ISO-8859 standard has been supplanted by the UTF-8 codeset.

Environment variables implement the locale codes. For example, **LANG=en_US.UTF-8** specifies that the computer uses US English with a UTF-8 encoding when displaying information. The following table lists the configurable environment variables:

Variable	Explanation

name	
LANG	Defines all locale settings at once, while allowing further individual customization via the **LC_*** settings. When LANG=C, programs display output without passing it through locale translations. This is helpful when the output is being corrupted by the locale and will help avoid some types of problems (e.g., when using pipelines and scripts that pass on a program's data to another program in binary form). Localization support is the responsibility of the program's author. Many programs only support one language or a small subset of languages.
LC_CTYPE	Defines the character handling properties for the computer. This determines whether characters are recognized as alphabetical, numeric, etc. This also determines the character set used (if applicable).
LC_MESSAGES	Specifies localizations for applications that use a message-based localization scheme.
LC_COLLATE	Defines the alphabetical ordering of strings. For example, the output of sorted directory listings.
LC_NUMERIC	Controls formatting for numeric values that are not monetary (e.g., which character to use as the thousands separator and the decimal separator).
LC_MONETARY	Identifies currency units and formatting of currency type and numeric values.
LC_TIME	Defines formatting for dates and times.
LC_PAPER	Designates the default paper size.
LC_NAME	Denotes personal name format (e.g., whether the surname comes first or last).
LC_ADDRESS	Specifies address formatting.
LC_TELEPHONE	Defines telephone number format.
LC_MEASUREMENT	Determines which measurement units are used.
LANGUAGE	Overrides the LC_MESSAGES settings.
LC_ALL	Sets all locales to the same setting. This is used for overriding all other settings.

The following table describes the commands and options used to configure locale settings:

Command	Description
locale	Displays the current locale settings for the computer. • **charmap** displays the character encoding. • **-a** lists all installed locales. • **-m** lists all installed character encoding options. The locale command is located in /usr/bin.
iconv	Converts encoding from one encoding type to another. • **-f** specifies the old encoding type. • **-t** specifies the new encoding type.

	• **-o** specifies the input and output file.

From <https://cdn.testout.com/client-v5-1-10-407/startlabsim.html?culture=en-us>

Extra

Monday, April 10, 2017 9:09 AM

yum install system-config-services

Chown - change owner user/group
Chmod - change permission
Useradd - add a new user
Groupadd -add a new group
Usermod - modify a user -1 or -l ?? Change
-r always goes recursive for following folder

3.1.2

Monday, February 27, 2017 10:00 AM

The boot process for a Linux computer includes the following general stages:

Stage	Process
BIOS	In the BIOS stage, BIOS is loaded and the system hardware is identified. The following steps take place: 1. Power is supplied to the processor. The processor is hard-coded to look at a special memory address for code to execute. 2. This memory address contains a pointer or jump program that instructs the processor where to find the BIOS program. 3. The processor loads the BIOS program. The first BIOS process to run is the power on self test (POST). 4. If the POST is successful, the BIOS identifies other system devices. It uses CMOS settings and information supplied by the devices themselves to identify and configure hardware devices. Plug and Play devices are allocated system resources. The system typically displays information about the keyboard, mouse, and IDE drives in the system. Following this summary, information about devices and system resources is displayed. 5. The BIOS then searches for a boot sector using the boot order specified in the CMOS.
Boot loader	During the boot loader stage, BIOS gives control to the boot loader program. The following steps take place: 1. BIOS searches the boot sector which contains a Master Boot Record (MBR). 2. BIOS loads the primary boot loader code from the MBR. 3. The primary boot loader does one of the following: • It examines the partition table marked as bootable, and then loads the boot sector from that partition. This boot sector contains a secondary boot loader, which locates an OS kernel. • It locates an OS kernel directly without using a secondary boot loader. 4. When the secondary boot loader is in RAM and executing, a splash screen is commonly displayed, and an optional initial RAM disk (e.g., initrd image) is loaded into memory. The initrd image: • Has root permissions that can be used to access the actual /root file system regardless of whether it exists on the local computer or an external device. Without the permissions, the computer could not access the file systems and read information that only exists on those file systems. • Is used to mount the actual file system and load the kernel into RAM. 5. With the images ready, the secondary boot loader invokes the kernel image.
OS Kernel	During this stage, the Linux kernel takes over. The kernel: 1. Resides in the /boot directory. 2. Initializes the hardware on the system. 3. Locates and loads the initrd script to access the linuxrc program which configures the operating system. 4. Dismounts and erases the RAM disk image. On older distributions, this is the initrd image. On newer distributions, this is the initramfs image.

5. Looks for new hardware and loads the drivers.
6. Mounts the root partition.
7. Loads and executes either the init (Initial) process (for older distributions) or the systemd process (for newer distributions). These processes then launch all other processes (either directly or indirectly) to finish booting the system.

   The init (Initial) or systemd processes are always assigned a process ID of 1 because they are always the first processes to run on the system.

This process has been used for decades. However, be aware that recent computer systems no longer use this process. Instead of using a traditional BIOS, these systems now use the *Unified Extensible Firmware Interface* (UEFI). Some of the features provided by UEFI include the following:

- It preserves several components from the traditional BIOS, including power management and a real time clock.
- It provides support for larger disks using *Globally Unique Identifier Partition Table* (GPT) partition scheme. The MBR partitioning scheme only supports four partitions per disk, with a maximum size of 2 TB per partition. UEFI supports a maximum partition size of 9.4 ZB (9.4 x 10^{21} bytes).
- It provides its own boot manager. This is a significant change. The old BIOS only had enough intelligence to load a single block from the storage device. This required a multi-stage boot process. UEFI has its own command interpreter and boot manager. You no longer need a dedicated boot loader as long as you place the operating system's bootable files into the UEFI system partition, which is formatted with the FAT32 file system.

As a result, UEFI has the intelligence to load an operating system on its own. To do this, UEFI reads the GUID partition table, which is located in the blocks immediately after block 0. The GUID partition table defines the layout of the partition table on the storage device. Using this information, the UEFI boot loader locates the *UEFI system partition* which contains the boot loader files for all operating systems that are installed on other partitions on the device. To boot Linux, you would use a UEFI-aware version of the GRUB bootloader and install its boot file (grub.efi) in the UEFI system partition.

The rest of the Linux boot process is the same as that used by older BIOS-based systems.

From <https://cdn.testout.com/client-v5-1-10-407/startlabsim.html?culture=en-us>

3.2.3

Wednesday, April 12, 2017 9:40 AM

GRUB2, the updated version of the Grand Unified Boot Loader (GRUB) utility, is any version of GRUB 1.98 or later. The following commands can be used to view the version of GRUB that has been installed. The actual command used will vary between distributions:

- **grub-install -v**
- **grub2-install -v**
- **grub2-install --version**

Earlier versions of GRUB are sometimes called *GRUB Legacy*.

Be aware of the following details about GRUB2:

- **/boot/grub/grub.cfg** or **/boot/grub2/grub.cfg** are the configuration files for GRUB2 (depending upon the distribution). These files are similar to the GRUB Legacy's **/boot/grub/menu.lst** or **/boot/grub/grub.conf** files.
- Depending on the distribution, the **update-grub** or **grub2-mkconfig** command generates the **/boot/grub2/grub.cfg** file. Specifically, these commands use the **/etc/default/grub** file and the scripts in the **/etc/grub.d/** directory to generate the **/boot/grub2/grub.cfg** configuration file.
- The **/etc/grub.d/** directory holds script files that are read when the **update-grub** or the **grub2-mkconfig** command is used.
- The **grub.cfg** file should not be edited directly. Instead, the appropriate configuration file below should be edited when a change needs to be made. Then run either the **update-grub** or **grub2-mkconfig** command.

  Important script files in the **/etc/grub.d/** directory include:

Script File	Description
00_header	Sets initial appearance items, such as the graphics mode, default selection, timeout, etc. These settings are typically imported from the **/etc/default/grub** file.
05 _debian_theme	Sets the GRUB2 background image, text colors, selection highlighting, and themes.
10_linux	Identifies all Linux kernels installed on the root device and creates corresponding GRUB2 menu entries for each one. This allows you to select which Linux kernel you want to load when you initially boot the system.
30_os-prober	Executes **os-prober** to search for other operating systems (such as Microsoft Windows) and automatically creates GRUB2 menu items for them.
40_custom	Allows for custom menu entries, which are imported directly into **/boot/grub/grub.cfg** without any changes.

The **/etc/default/grub** file is the primary configuration file for changing menu display settings. The following table describes several common options in the configuration file:

Option	Description	Examples
GRUB_DEFAULT	Sets the default menu entry. Typical entries include: • Numeric (e.g., 0, 1, 2, etc.)	**GRUB_DEFAULT=0** sets the first menu entry as the default.

	• Complete menu entry quotation (e.g., "Ubuntu, Linux 2.6.31-9-generic")	**GRUB_DEFAULT="Ubuntu, Linux 2.6.31-9-generic"** sets a menu entry as the default.
GRUB_SAVEDEFAULT	Sets the last selected OS from the menu as the default OS on the next boot. **GRUB_DEFAULT=saved**is also required for this option to work correctly.	**GRUB_SAVEDEFAULT=true** sets the last selected OS from the menu as the default OS on the next boot.
GRUB_HIDDEN_TIMEOUT	Determines how long a blank screen will be displayed. While the screen is blank, the user can press the Shift key to display the GRUB2 menu. Options include: • **0** disables this functionality. • *X* (an integer value) pauses and shows a blank screen for *X* seconds. • (null) uses the value specified in the GRUB_TIMEOUT entry.	**GRUB_HIDDEN_TIMEOUT=0** disables this functionality. **GRUB_HIDDEN_TIMEOUT=3** displays a blank screen for 3 seconds and then boots to the default OS if there is no user interaction.
GRUB_HIDDEN_TIMEOUT_QUIET	Works in conjunction with the GRUB_HIDDEN_TIMEOUT parameter. It displays a counter (countdown) while the screen is blank. Options include: • **true** does not display a counter. • **false** displays the counter for the duration specified in the GRUB_HIDDEN_TIMEOUT entry.	**GRUB_HIDDEN_TIMEOUT_QUIET=true** does not display a counter.
GRUB_TIMEOUT	Determines how long to wait for user interaction before booting the default operating system. Options include: • *X* (an integer value of 1 or higher) sets the display duration in seconds. • **-1** causes the menu to display until the user makes a selection. The GRUB2 menu is hidden by default unless another OS is detected by the system.	**GRUB_TIMEOUT=4** causes the menu to display for four seconds and then boots into the default operating system. **GRUB_TIMEOUT=-1** causes the menu to display until the user makes a selection.
GRUB_CMDLINE_LINUX	Passes options to the kernel. With the GRUB Legacy bootloader, this was done by adding options to the end of the kernel line. In GRUB2, this is done using the GRUB_CMDLINE_LINUX parameter.	
GRUB_GFXMODE	Sets the resolution of the graphical GRUB2 menu. Multiple resolutions may be specified if they are separated by commas.	**GRUB_GFXMODE=1440x900x24** sets the screen resolution to 1440 x 900 with a color depth of 24-bit.
GRUB_BACKGROUND	Sets the background image during the GRUB2 menu display. The full path should be used. Must be in the PNG, TGA, or JPG/JPEG file formats.	**GRUB_BACKGROUND=/usr/share/images/back.png** displays back.png as the background image.
GRUB_DISA	Enables and disables the os-prober check of other	**GRUB_DISABLE_OS_PROBER**

BLE_OS_PROBER	partitions for operating systems, including Windows, Linux, etc., during execution of the **update-grub** command. If the os-prober is enabled, operating systems found will be placed in the GRUB2 menu.	**=true** disables the os-prober. **GRUB_DISABLE_OS_PROBER=false** enables the os-prober and will add found operating systems to the GRUB2 menu.

3.3.5

Wednesday, April 19, 2017 9:42 AM

The systemd daemon uses boot targets to set the system state. A *boot target* represents one of several different modes that the Linux system can be configured to run in.

Earlier versions of Linux that were based on the init daemon used runlevels instead of boot targets to set the system state.

The default boot target specifies the state that your system will boot into each time it is turned on. In addition, the system state can be dynamically switched between boot targets while the operating system is running. Each boot target is represented by a file (called a *target unit*) in **/usr/lib/systemd/system**:

Target File	Description
poweroff.target	Halts the system.
rescue.target	Configures the system to run in single user mode with a text-based user interface. This target sets up a base system and opens a rescue shell for troubleshooting system problems.
multi-user.target	Configures the system to run in multi-user mode with a text-based user interface. This target is commonly used as the default mode for server systems.
graphical.target	Configures the system to run in multi-user mode with a graphical user interface. This target provides all the services of the multi-user target with the addition of a graphical user interface. This target is commonly used as the default mode for desktop systems.
reboot.target	Reboots the system.
emergency.target	Opens a minimal emergency shell for troubleshooting serious system problems.

Because init-based distributions have been around for decades, systemd provides additional boot targets that use runlevel-like names to help ease the transition from init to systemd:

Target File	Description
runlevel0.target	Equivalent to **poweroff.target**
runlevel1.target	Equivalent to **rescue.target**
runlevel2.target **runlevel3.target** **runlevel4.target**	Equivalent to **multi-user.target**
runlevel5.target	Equivalent to **graphical.target**
runlevel6.target	Equivalent to **reboot.target**

The **systemctl** command is used to manage boot targets:

Command	Description	Example

systemctl isolate *boot_target*	Changes the system state to the specified boot target. Changing boot targets with the systemctl command only changes the current system state. If the system is restarted, it will revert back to the default boot target.	Either of the following commands can be used to switch the system to multi-user mode with a graphical user interface: • **systemctl isolate runlevel 5.target** • **systemctl isolate graphical.target**
systemctl get-default	Displays the current boot target.	**systemctl get-default**
systemctl set-default *boot_target*	Sets the default boot target, which is identified by the**/etc/systemd/system/default.target** file. This file is a symbolic link that points to a target file in /usr/lib/systemd/system that should be used by default when the system starts. This command modifies the target file that the default.target symbolic link points to.	Either of the following commands can be used to set the default boot target to graphical mode: • **systemctl set-default graphical.target** • **systemctl set-default runlevel 5.target**

On older Linux distributions, the **init** command was used to manage runlevels and system daemons. The **/etc/inittab** file was used to set the default runlevel.

3.4.5

Wednesday, April 19, 2017 9:57 AM

On a systemd-base Linux distribution, the **systemctl** command is used to stop, start, restart, or view the status of services on the system:

Use	To	Example
systemctl start *service*.service	Start a daemon.	**systemctl start nfs.service** starts the NFS daemon.
systemctl stop *service*.service	Stop a daemon.	**systemctl stop nfs.service** stops the NFS daemon.
systemctl restart *service*.service	Restart a daemon.	**systemctl restart nfs.service** stops and then restarts the NFS daemon.
systemctl reload *service*.service	Reload a daemon's configuration without actually stopping the daemon itself.	**systemctl reload nfs.service** reloads the NFS daemon's configuration file without stopping the daemon itself.
systemctl status *service*.service	Display a daemon's status.	**systemctl status nfs.service** displays the current status of the NFS daemon.
systemctl enable *service*.service	Automatically start a daemon when the system starts.	**systemctl enable nfs.service** automatically starts the NFS daemon every time the system boots.
systemctl disable *service*.service	Prevent a daemon from automatically starting when the system starts.	**systemctl disable nfs.service** disables the NFS daemon so that it does not automatically start when the system boots.
systemctl is-enabled *service*.service	See if a daemon is configured to automatically start on system boot.	**systemctl is-enabled nfs.service** displays whether or not the NFS daemon will be automatically started when the system boots.
systemctl mask *service*.service	Prevent a daemon from starting at all, either automatically or manually from the shell prompt.	**systemctl mask nfs.service** prevents the NFS daemon from starting at all.
systemctl unmask *service*.service	Unmask a previously masked daemon. This allows the daemon to be started either manually or automatically.	**systemctl unmask nfs.service** unmasks the NFS daemon.

Earlier versions of Linux that were based on the init daemon used *init scripts* along with the **chkconfig**, **insserv**, **rc**, and **service** commands to manage system services.

3.5.4

Wednesday, April 19, 2017 10:06 AM

Turning off the power without executing the proper shutdown procedure to a computer can result in data loss and file system corruption. Linux provides several different shutdown options. The table below shows common commands for shutting down the system:

Use	To	Examples
shutdown -h now **halt** **init 0** **systemctl isolate poweroff.target** **systemctl isolate runlevel0.target**	Shut the system down immediately. For the shutdown command: • **-h** specifies that the system halt or power off after shutdown. • **now** forces the system to shut down without a delay. Any of these commands shuts the system down immediately.	
shutdown -r now **reboot** **init 6** **systemctl isolate reboot.target** **systemctl isolate runlevel1.target**	Shut the system down immediately and reboot. Any of these commands reboots the system immediately.	
shutdown -h *time message* **shutdown -r** *time message*	Shut the system down in the designated amount of time and send a message.	**shutdown -h +5 System is going down** sends a message and shuts the system down in five minutes. **shutdown -h 22:00** shuts the system down at 10:00 pm. **shutdown -r +15** reboots the system in 15 minutes. **shutdown -r 24:00 System is going down at Midnight** sends a message and reboots the system at midnight.
shutdown -c **Ctrl+c**	Cancel a pending shutdown.	
shutdown -rf *time*	Reboots the system and skips the **fsck** utility on reboot. The **-f** parameter stands for reboot fast.	**shutdown -rf** reboots the system and skips **fsck**. **shutdown -r +15** reboots the system in 15 minutes and uses **fsck**.

shutdown -k*message*	Sends a warning message, but does not shut down the system.	**shutdown -k Please log out of the system** sends the message to logged in users, but does not actually shut the system down.

When using the **shutdown** command to shut down the computer, the system does the following:

1. Sends a SIGTERM message to open programs to allow them to close.
2. Notifies logged on users that the shutdown process has initiated and the length of time before shut down.
3. Blocks users from logging into the system.
4. Shuts down all running processes and then the system.

4.1.2

Friday, April 21, 2017 9:18 AM

The X Window System provides the graphical user interface (GUI) on Linux systems. It is a modular system that gives administrators control over each component. The X Window System is made up of the following components:

Component Description

X server The X server is the main component of the X Window System. The X server:

Manages input devices such as the mouse and keyboard and controls output to monitors and printers.

Is networked, which means its output can be displayed locally or sent over the network to other computers.

Uses the DISPLAY environment variable to control where the output is sent. Setting DISPLAY to 0:0 will send the X server output to the monitor on the local system.

Has two common implementations.

X.Org (also known as X11) is the most commonly used X server in modern Linux distributions.

XFree86 is an earlier X server implementation that was used by default in most Linux distributions up until 2005.

Window manager A window manager controls the placement and appearance of windows on a Linux computer (such as moving, hiding, resizing, or closing), as well as controlling what they display. Most distributions come with multiple window managers, such as the following:

Enlightenment

Sawfish

Flexible Virtual Window Manager (FVWM)

KDE Window Manager (KWin)

Tab Window Manager (twm)

Window Maker (Wmaker)

Desktop environment A desktop environment controls the desktop features, including desktop menus, screensavers, wallpapers, desktop icons, and taskbars. The two most common desktop environments are:

GNU Network Object Model Environment (GNOME)

Kool Desktop Environment (KDE)

X font server An X font server is an optional server on the network that manages fonts for client computers. X font servers were developed when client systems were not powerful enough to handle font rendering without using significant CPU cycles, so a single networked server provided font rendering for the client computers. Because clients can easily render fonts now, X font servers are rarely used.

The fontpath section of the xorg.conf file controls where a computer looks for fonts.

An entry similar to FontPath "unix:/7100" specifies that the local computer is the font server.

An entry similar to FontPath "tcp/fonts.acme-u.edu" gives the name of a networked font server.

Many Linux distributions have font tools and systems that are unique to that distribution and may handle fonts differently. Using these tools is the recommended way to handle fonts.

The X server configuration is stored in one of the following text files in the /etc directory:

When using X.Org on a Linux distribution that uses the init daemon, your configuration settings are saved in /etc/X11/xorg.conf.

When using XFree86, the configuration settings are saved in /etc/X11/XF86Config.

If the distribution is based on systemd, the X11 configuration is stored in a series of configuration files located in /etc/X11/xorg.conf.d. Separate configuration files are used in this directory to configure different aspects of the video system, such as 10-evdev.conf, 50-device.conf, 50-monitor.conf, and 50-screen.conf. The syntax within these files is similar to that used in the xorg.conf file.

The X server configuration is stored in one or more text files that can be modified with a text editor. However, do not edit these files unless absolutely necessary. The configuration files are very complex. A

single mistake can cause hardware damage. Instead, use the configuration utility that came with the distribution to configure X server settings. Below are some commonly used configuration utilities:
SaX2 is used to configure the systemâ€™s video adapter, monitor, resolution, and color depth. SaX2 was used for many years; however, most modern distributions no longer use SaX2 to configure the video system.
xorgconfig is a utility used from the command line to configure the video board, monitor, resolution, and color depth.
xorgcfg is a graphical version of the xorgconfig utility that is loaded by entering xorgcfg from the shell prompt. This utility allows you to configure the monitor, video board, keyboard, and mouse. It is included with the X.Org X server.
Xorg -configure is a command used to automatically detect system hardware and create the configuration file /root/xorg.conf.new. The configuration can then be tested before committing it by entering X â€“config /root/xorg.conf.new at the shell prompt. If everything works correctly, renaming the file to /etc/X11/xorg.conf enables the new configuration.
xdpyinfo is a command used to display the capabilities of the X server, such as the different screen modes available.
xwininfo is a command that displays information about open windows on the desktop.
xvidtune is run from the shell prompt and is used to fine-tune the X server configuration. This utility can be used to customize the monitorâ€™s horizontal and vertical synchronization.
Using sync parameters that are beyond the monitorâ€™s capabilities will eventually damage it.
If the distribution uses the XFree86 X server, use the following configuration utilities:

XFree-86 â€“configure detects system hardware and creates the configuration file /etc/X11/XF86Config.new. Enable the configuration by renaming the file to /etc/X11/XF86Config.
xf86config is a text-based X server configuration utility.
xf86cfg is a graphical X server configuration utility.
After making any changes to the X server configuration, do the following:

Test the new settings to make sure they work.
Restart the X server to apply the changes. The easiest way to do this is to simply log out and then log back in. Rebooting the system will also accomplish the same task.
If something is misconfigured and the X server software is hung, kill it by pressing Ctrl+Alt+Backspace.
To start the Linux GUI from inside a text-based environment (such as runlevel 3 or multi-user.target), enter startx or X at the shell prompt.
Because the X server works directly with the video card and monitor, configuring it correctly is critical. An incorrect configuration can cause hardware damage.

From <http://www.sildom.me/SS/4.1.2>

4.2.4

Friday, April 21, 2017 9:18 AM

A display manager is an application that works with the X server to provide window control elements, manage virtual desktops, and provide window framing (such as resizing a window or moving it to a different location on the screen). When the display manager is loaded, it starts the X server. Then the display manager presents the user with a login screen. Commonly used Linux display managers include:
The X Window Display Manager (xdm) is a very early display manager. It provides minimal functionality and isn't commonly implemented any more.
The KDE Display Manager (kdm) is used by Linux systems that use the KDE desktop environment.
The GNOME Display Manager (gdm) is used by Linux systems that use the GNOME desktop environment.
The LightDM display manager is the default display manager used by the Ubuntu distribution. It uses the concept of modular front-ends (called greeters) to create login screens. The most popular greeter is the Unity greeter. However, there are many other greeters that you can use.
The display manager can be customized by editing the appropriate configuration file:

Display Manager	Configuration File
xdm	/etc/X11/xdm/xdm-config
LightDM	The LightDM display manager is configured using several different files: /usr/share/lightdm/lightdm.conf.d /etc/lightdm/lightdm.conf.d /etc/lightdm/lightdm.conf
kdm	/etc/kde/kdm or /etc/X11/ kdm
gdm	/etc/X11/gdm

The default LightDM configuration is stored in the configuration files in /usr/share/lightdm/lightdm.conf.d/. These files are generated by the system and should not be edited directly. Instead, to customize the LightDM display manager, create override files containing just the desired changes in /etc/lightdm/lightdm.conf.d/. If there are duplicate configuration settings in multiple files, the setting in the last file processed is applied and the others are ignored.
Alternatively, customized settings can be entered in the /etc/lightdm/lightdm.conf file.
LightDM uses a greeter, which provides an interface that allows users to interact with LightDM. On Ubuntu, the Unity greeter is used by default. However, there are many other greeters that can be used. In addition, the way each greeter works can be customized:

Configuration Task	Configuration File	Setting
Use a different greeter	Create an override file in /etc/lightdm/lightdm.conf.d/.	[SeatDefaults] greeter-session=file_name Replace file_name with the name of the greeter file to be used. Greeters are stored in /usr/share/xgreeters and have an extension of .desktop.
Disable guest logins	Create an override file in /etc/lightdm/lightdm.conf.d/.	[SeatDefaults] allow-guest=false
Hide the user list	Create an override file in /etc/lightdm/lightdm.conf.d/.	[SeatDefaults] greeter-hide-users=true
Allow manual logins	Create an override file in /etc/lightdm/lightdm.conf.d/.	[SeatDefaults] greeter-show-manual-login=true
Change the default session type	Create an override file in /etc/lightdm/lightdm.conf.d/.	[SeatDefaults] user-session=file_name

Session files are located in /usr/share/xsessions and have an extension of .desktop.

From <http://www.sildom.me/SS/4.2.4>

4.3.3

Monday, February 27, 2017 10:01 AM

Accessibility options (also known as Assistive Technologies) allow people with tactile, audible, and visual impairments to use Linux systems.

The keyboard accessibility options (also known as AccessX) can filter out accidental key presses and allow users to use shortcut keys (such as Ctrl+c) without having to hold down multiple keys at the same time. The following table describes keyboard accessibility options:

Feature	Description
Sticky keys	Sticky keys causes a keyboard modifier (e.g., Ctrl, Alt, or Shift) to "stick" when pressed. This affects the next regular key to be pressed, even after the release of the sticky key. This is useful for users who have difficulty pressing multiple keys at the same time.
Mouse Keys	Mouse keys controls the mouse pointer with the number keypad.
Slow keys	Slow keys requires a key to be pressed for a specified time period before acceptance. This is useful for individuals who tend to accidentally press keys.
Toggle keys	Toggle keys associates sounds when the Caps Lock and/or Num Lock is on.
Repeat rate	Repeat rate affects how quickly the action associated with the key is repeatedly performed when the key is pressed and held down. For example, if you press and hold a character key, the character is typed repeatedly according to the repeat rate.
Bounce keys	Bounce keys ignores fast key presses of the same key, compensating for when users accidentally press a single key multiple times.

Assistive Technologies also provide mouse accessibility options for physically impaired users, such as the following:

Feature	Description
Simulated secondary click	The simulated secondary click option allows you to send a double-click by simply holding down the primary mouse button for a specified amount of time.
Dwell click	The dwell click option sends a mouse click whenever the mouse pointer stops moving for a specified amount of time.
Mouse gesture	A mouse gesture allows users to configure Linux to complete a specified task when the mouse is moved in a certain pattern. You can configure an action for each of the following gestures:

Single click
Double click
Drag click
Secondary click

For each gesture, you can configure one of the following actions:

Move left
Move right
Move up
Move down

Visual accessibility options include the following:

Feature	Description
Onscreen keyboard	An onscreen keyboard displays an image of a keyboard which allows a user to use a mouse to select keys as if they were pressed on a real keyboard. The GNOME On-Screen Keyboard (GOK) provides the onscreen keyboard, but also helps users navigate the GNOME desktop with the use of alternative input methods, such as:

Blowing and sipping to activate a pneumatic switch
Blinking an eye or a directed gaze with an eye tracking system
Moving the head
Contracting muscles or moving limbs

Feature	Description
Mouse gesture	A mouse gesture allows users to configure Linux to complete a specified task when the mouse is moved in a certain pattern.

Screen reader	A screen reader reads the text on a screen including menu and button text. Popular screen readers on Linux systems include the following:
Orca is a free, open-source scriptable screen reader that works with the GNOME desktop.
Emacspeak is a free screen reader and is often bundled with text editors.
Screen magnifier	A screen magnifier creates an enlarged view of the area around the mouse pointer by default.
Braille devices	Linux can use the following Braille hardware devices:
A Braille display is a special type of computer monitor which creates a tactile display of textual information. Many Linux text-mode applications manage Braille display with no configuration changes.
A Braille Embosser prints a hard copy of a text document using embossed Braille characters.
BRLTTY provides a Linux daemon that redirects text-mode output to a Braille device.
Desktop themes	A desktop theme is a preset package containing graphical appearance details. Desktop themes of an assistive nature include the following:
A high contrast theme that increases the background and text color contrast to improve readability.
A large print theme that enlarges the onscreen text to improve readability.

From <http://www.sildom.me/SS/4.3.3.txt>

6.1.4

Friday, April 21, 2017 10:20 PM

User accounts control the ability to log on to a system, access resources, and perform certain actions. *Groups* provide a means of grouping users for administrative purposes such as assigning permissions to files. Be aware of the following types of users and groups:

Type	Description
Standard user	Standard user accounts can log into the system. Standard user accounts: • Have friendly usernames (such as Mary or bkaun). An administrator must create the usernames. • Have an ID of 500 or more (on some distributions) or 1000 or more (on other distributions). The ID is automatically assigned by the system when the account is created.
System user	System user accounts are created by default during the Linux installation and are used by the system for specific roles. System user accounts: • Have names that correspond with their roles, such as ftp and mail. • Cannot be used to log into the system. • Have an ID of 500 or less (on some distributions), or 1000 or less (on other distributions). The ID is automatically assigned by the system when the account is created. The root user account is created by default and has a UID of 0; however, it can be used to log into a system and perform tasks.
Primary group	Primary groups (also called the private group) are created by default on most Linux distributions when a standard user is created and are used to manage access to files and directories. Primary groups: • Have the corresponding user as the only member. • Are automatically assigned as the owner of files and directories when they are created in the file system. • Are similar to any other group. The only difference is that the group is identified as the primary group in the user account's configuration.
Secondary group	Secondary groups are also used to manage access to files and directories. Secondary groups: • Are not automatically assigned user accounts as members. • Receive their membership as assigned by the system administrator.

The user and group databases are stored in the following files:

File	Description
/etc/passwd	The **/etc/passwd** file holds user account information. Be aware of the following details: • Each entry identifies a user account. • Each entry contains multiple fields, with each field separated by a colon. The following line is a sample entry in the **/etc/passwd** file: `pclark:x:501:501:Petunia Clark:/home/pclark:/bin/bash` The fields within this line are as follows: 1. User account name. 2. Password. An x in the field indicates passwords are stored in

	the **/etc/shadow** file. 3. User ID number. 4. Primary group ID number (also known as the GID). 5. Description field. This field is typically used for the user's full name. 6. Path to the home directory. 7. Path to the default shell.
/etc/shadow	The **/etc/shadow** file holds passwords and password expiration information for user accounts. Be aware of the following details: • Using the **/etc/shadow** file to separate usernames from passwords increases the security of the user passwords. • Like the **/etc/passwd** file, each entry corresponds to a user account and each entry contains multiple fields, with each field separated by a colon. The following line is a sample entry in the **/etc/shadow** file: `pclark:$ab7Y56gu9bs:12567:0:99999:7:::` The fields within this line are as follows: 1. User account name. 2. Password. • $ preceding the password identifies the password as an encrypted entry. • ! or !! indicates that the account is locked and cannot be used to log in. • * indicates a system account entry and cannot be used to log in. 3. Last change. The date of the most recent password change, measured in the number of days since 1 January 1970. 4. Minimum password age. The minimum number of days the user must wait before changing the password. 5. Maximum password age. The maximum number of days between password changes. 6. Password change warning. The number of days a user is warned before the password must be changed. 7. Grace logins. The number of days the user can log in without changing the password. 8. Disable time. The number of days since 1 January 1970, after which the account will be disabled.
/etc/group	The **/etc/group** file holds group information including the group name, GID, and group membership information. Be aware of the following details: • Each entry identifies a group. • Each entry contains multiple fields, with each field separated by a colon. The following line is a sample entry in the **/etc/group** file: `sales:x:510:pclark,mmckay,hsamson` The fields within this line are as follows: 1. Group name. 2. Group password. An x indicates the group passwords are contained in the **/etc/gshadow** file. 3. Group ID. 4. Group members. Contains a comma-separated list of user accounts that are members of the group.
/etc/gshadow	The **/etc/gshadow** file holds passwords for groups. Be aware of the following details: • Like the /etc/group file, each line corresponds to a group. • Each line consists of fields separated by colons.

	The following line is a sample entry in the **/etc/gshadow** file: `sales:!:pclark:pclark,mmckay,hsamson` The fields within this line are as follows: 1. Group name. 2. Group password. The group password allows users to add themselves as members of the account. • If the field contains a single exclamation point (!), the group account cannot be accessed using the password. • If the field contains a double exclamation point (!!), no password has been assigned to the group account (and it cannot be accessed using the password). • If there is no value, only group members can log in to the group account. 3. Administrators. Contains a comma-separated list of users who have authorization to administer the account. 4. Group members. Contains a comma-separated list of user accounts that are members of the group.

Additional commands for managing file entries include:

- The **pwck** command verifies the entries in the **/etc/passwd** and **/etc/shadow** files. Errors are displayed on the screen, and entries may be deleted to solve the errors.
- The **pwconv** command synchronizes the entries in the **/etc/passwd** and **/etc/shadow** files.

From <https://cdn.testout.com/client-v5-1-10-414/startlabsim.html>

6.2.4

Friday, April 21, 2017 10:21 PM

Be aware of the following configuration files when managing user accounts:

File	Description
/etc/default/useradd	The **/etc/default/useradd** file contains default values used by the **useradd** utility when creating a user account, including: • Group ID • Home directory • Account expiration • Default shell • Secondary group membership • Skeleton directory
/etc/login.defs	The **/etc/login.defs** file defines: • Values used to define allowed group and user ID numbers. • Protocols to be used for password encryption in the shadow file. • Password aging values for user accounts. • The path to the default mailbox directory. • Whether a home directory should be created by default.
/etc/skel	The **/etc/skel** directory contains a set of configuration file templates that are copied into a new user's home directory when it is created, including the following files: • **.bashrc** • **.bash_logout** • **.bash_profile** • **.kshrc**

Although it is possible to edit the **/etc/passwd** and **/etc/shadow** files manually to manage user accounts, doing so can disable your system. Instead, use the following commands to manage user accounts:

Use	To	Example
useradd	Create a user account. The following options override the settings found in **/etc/default/useradd**: • **-c** adds text for the account in the description field of **/etc/passwd**. This is commonly used to specify the user's full name. • **-d** assigns an absolute pathname to a custom home directory location. • **-D** displays the default values specified in the **/etc/default/useradd** file. • **-e** specifies the date on which the user account will be disabled. • **-f** specifies the number of days after a password expires until the account is permanently disabled.	**useradd pmaxwell** creates the *pmaxwell* user account**.** **useradd -c "Paul Morril" pmorril** creates the *pmorril* account with a comment**.** **useradd -d /tmpusr/sales1 sales1** creates the *sales1*user account with home directory located at*/tmpusr/sales1*. **useradd -u 789 dphilips** creates the *dphilips* account with user ID *789*.

	• **-g** defines the primary group membership. • **-G** defines the secondary group membership. • **-M** does not create the user's home directory. • **-m** creates the user's home directory (if it does not exist). • **-n, N** does not create a group with the same name as the user (Red Hat and Fedora respectively). • **-p** defines the encrypted password. • **-r** specifies that the user account is a system user. • **-s** defines the default shell. • **-u** assigns the user a custom UID. This is useful when assigning ownership of files and directories to a different user.	
passwd	Assign or change a password for a user. • **passwd** (without a username or options) changes the current user's password. • Users can change their own passwords. The root user can execute all other **passwd** commands. Be aware of the following options: • **-S** *username* displays the status of the user account. • LK indicates that the user account is locked. • PS indicates that the user account has a password. • **-l** disables (locks) an account. This command inserts a !! before the password in the **/etc/shadow** file, effectively disabling the account. • **-u** enables (unlocks) an account. • **-d** removes the password from an account. • **-n** sets the minimum number of days a password exists before it can be changed. • **-x** *s*ets the number of days before a user must change the password (password expiration time). • **-w** sets the number of days before the password expires that the user is warned. • **-i** sets the number of days following the password expiration that the account will be disabled.	**passwd jsmith** changes the password for the jsmith account. **passwd -d** removes the password from an account. **passwd -d jsmith** removes the password from the jsmith account. **passwd -x 40 jsmith** requires jsmith to change his password every 40 days. **passwd -n 10 jsmith** means that jsmith cannot change his password for 10 days following the most recent change. **passwd -w 2 jsmith** means that jsmith will be warned 2 days before his password expires. **passwd -i 7 jsmith** disables the jsmith account after 7 days if the password is not changed. **passwd -l jsmith** locks the jsmith account. **passwd -u jsmith** unlocks the jsmith account.
usermod	Modify an existing user account. **usermod** uses several of the same switches as **useradd**. Be aware of the	**usermod -c "Paul Morril" pmorril** changes the comment field for user pmorril.

	following switches: • **-a** appends the user to the supplementary groups specified with the **-G** option. • **-c** changes the description for the account. This is usually used to modify the user's full name. • **-d home_dir** assigns the user a new home directory. If -d is used with the -m option, the contents of the user's current home directory will be moved to the new home directory. • **-e *date*** specifies the date when the account will be disabled. • **-f** specifies the number of days after a password expires until the account is permanently disabled. • **-g** specifies the primary group membership. • **-G** specifies the secondary group membership. This option is usually used in conjunction with the **-a** option. If you don't use the **-a** option, then **-G** will overwrite all existing supplementary group memberships. • **-l** renames a user account. When renaming the account: • Use **-d** to rename the home directory. • Use **-m** to copy all files from the existing home directory to the new home directory. • **-L** locks the user account. This command inserts a ! before the password in the **/etc/shadow** file, effectively disabling the account. • **-m** moves the contents of the user's home directory to the new location specified by the -d option. • **-p** *password* assigns the specified encrypted password to the account. • **-s** *shell* sets the user's default login shell. • **-u** *UID* assigns a new user ID number. • **-U** unlocks the user account.	**usermod -l esmith -d /home/esmith -m ejones** renames the ejones account to esmith, renames the home directory, and moves the old home directory contents to the new location. **usermod -s /bin/tsch esmith** points the shell for esmith to /bin/tsch. **usermod -U esmith** unlocks the esmith account.
userdel	Remove the user from the system. Be aware of the following options: • **userdel *username*** (without options) removes the user account. • **-r** removes the user's home directory. • **-f** forces the removal of the user account even when the user is logged into the system.	**userdel pmaxwell** deletes the pmaxwell account while leaving the home directory on the hard drive. **userdel -r pmorril** removes both the account and the home directory.

From <https://cdn.testout.com/client-v5-1-10-414/startlabsim.html>

6.3.3

Monday, April 24, 2017 10:43 AM

Use the following commands and options to manage group accounts and group membership:

Use	To	Example
groupadd	Create a new group. The following options override the settings found in **/etc/login.defs**: • **-g** defines the group ID (GID). • **-p** defines the group password. • **-r** creates a system group.	**groupadd sales** creates the sales group.
groupmod	Modify a group definition. **groupmod** options include: • **-n** changes the name of a group. • **-A** adds specified users from the group (not available on all distributions) • **-R** removes specified users from the group (not available on all distributions)	**groupmod -n sales2 sales** renames the sales group to sales2. **groupmod -R rsem sales** removes the rsem account from the sales group.
groupdel	Delete a group.	**groupdel mktg** deletes the mktg group.
gpasswd	Change a group password. • ***groupname*** prompts for a new password. • **-r** removes a group password.	**gpasswd sales** prompts for a new group password.
newgrp	Log in to a new group with the group password.	**newgrp sales** prompts for the password for the sales group before logging in.
usermod	Modify group membership for the user account. Be aware of the following options: • **-g** assigns a user to a primary group. • **-G** assigns a user to a secondary group (or groups). Follow the command with a comma-separated list of groups. If the user already belongs to any secondary groups, the user will be removed from those groups if the groups are not in the list. • **-aG** assigns a user to a secondary group (or groups) by appending them to any groups the user already belongs to. Follow the command with a comma-separated list of groups. • **-G ""** Removes the user from all secondary group memberships. Do not include a space between the quotes.	**useradd -g pmaxwell pmaxwell** assigns primary group membership for user pmaxwell to the pmaxwell group. **usermod -G sales,mktg pmorril** removes all existing secondary group assignments for pmorril and makes the user account a member of the sales and mktg groups. **usermod -aG acct,prod pmorril** keeps existing secondary group assignments for pmorril intact and makes the user account a member of the acct and prod groups. **usermod -G "" pmaxwell** removes the pmaxwell from all groups.
groups	Display the primary and secondary group	**groups pmaxwell** displays group

	membership for the specified user account.	membership for the pmaxwell account.

The command options listed here are not applicable to every distribution of Linux. Consult the man pages for the options that are supported by the Linux distribution you are using.

From <https://cdn.testout.com/client-v5-1-10-414/startlabsim.html>

7.1.4

Monday, April 24, 2017 10:43 AM

The *master boot record* (MBR) partition format has been used by many operating systems, including Linux, since the 1980s. Because of its age, the MBR partition format has many limitations:

- The master boot record must be installed in the first 512 bytes of the hard disk.
- Only four standard partitions can be created on a storage device.
- The default block size of 512 bytes limits partitions to a maximum size of 2 TB.

Many workarounds have been implemented over the years to address these issues:

- *Logical Block Addressing* (LBA) allows the use of larger hard disks.
- Use of 4,096 byte sectors increases the maximum partition size on a disk.
- Extended partitions can contain many logical partitions.

A *partition* is a logical division of a storage device associated with a hard disk drive. A storage device using an MBR can have a single partition or multiple partitions. The most common partitioning scheme divides a disk into two different partition types:

Type	Description
Primary	A *primary* partition is used to store data as well as the operating system. Primary partitions: • Can hold operating system boot files. • Cannot be further subdivided into logical drives. • Can be formatted with a file system. There can be a maximum of four primary partitions or three primary partitions and one extended partition on a single hard disk drive.
Extended	An *extended* partition is an optional partition that contains logical partitions. Because an operating system cannot be booted from a logical partition from within an extended partition, this type of partition is not bootable. Extended partitions: • Can be further subdivided into an unlimited number of logical partitions. • Cannot be directly formatted with a file system. However, logical partitions within an extended partition can be formatted with a file system. Only one extended partition can exist on a single hard disk drive.

Use the following tools to create and manage partitions:

Tool	Description
fdisk	The fdisk utility is used to manage partitions on a hard disk. Be aware of the following fdisk characteristics: • A beginning/ending sector or size is requested when creating a partition. The size is indicated using K (kilobytes), M (megabytes), G (gigabytes), or T (terabytes). • It uses hexadecimal codes to determine the partition type. Common hexadecimal codes include: • 0x82 (Linux swap) • 0x83 (Linux partition) • 0x85 (Linux extended partition) • 0x8e (Linux LVM partition) • Using the **-l** option displays the current partition configuration on the system. Type **fdisk *[device_name]*** at the command prompt to enter the fdisk utility. Be

	aware of the following options within the utility: • **l** lists the partition types supported. • **m** displays the help screen. • **n** creates a new partition. • **p** displays the partition table for that device. • **q** exits fdisk without saving changes. • **w** writes the partition table to disk (i.e., saves the file) and exits the fdisk utility. • **d** deletes a partition.
partprobe	Requests that the operating system re-read the partition table. The operating system kernel reads the partition table and recognizes the table changes.

From <https://cdn.testout.com/client-v5-1-10-414/startlabsim.html>

7.1.5

Monday, April 24, 2017 10:44 AM

In a modern Linux system, there may be many different types of storage devices implemented:

Storage Device Type	Description
Hard disk drives(HDDs)	For decades, hard disk drives have been the primary type of long-term storage used in desktop and server systems. Hard disk drives magnetically store information using spinning aluminum disks called *platters*. Each platter is coated with a magnetic surface material that allows the hard disk's read/write *heads* to store and retrieve information to and from the drive. The faster the disk's platters spin, the faster data can be accessed. A hard disk drive identifies where data can be stored on its platters using several parameters that are collectively called the drive's *geometry*. The following parameters are used by the storage device interface to determine how the drive is accessed and where data can be stored: • **Heads** specifies to the number of read/write heads in the drive. • **Cylinders** specifies the number of concentric parallel tracks on all sides of all platters in the hard disk drive. • **Sectors Per Track** specifies the number of wedge-shaped areas the platters have been divided into. Hard disk drives are connected to the system motherboard using a storage interface. The interface is commonly integrated within the motherboard itself. However, it may also be implemented using an expansion card installed in an expansion slot. In a modern desktop computer system, the following storage interfaces may be used: • Serial ATA (SATA) • Small Computer System Interface (SCSI) • Parallel ATA (PATA) (This interface is obsolete) Hard disks provide several advantages, including the following: • They can store a large amount of data. • They provide reasonably fast access speeds. • The store data at a relatively low cost per megabyte. Hard disks also have several disadvantages, including the following: • Hard disks wear out over time because they are mechanical devices that contain moving parts. • Hard disks are vulnerable to physical damage. For example, dropping a hard drive while it is spinning can cause the read/write heads to dig into the platter, destroying any data stored there.
Solid state drives (SSDs)	A solid-state drive is a storage device that functions much like a hard disk drive, using the same block-based I/O operations. However, instead of aluminum platters, SSDs use flash memory to store data. SSDs typically provide storage capacity comparable to that of a small hard drive. SSDs are beginning to replace standard hard disk drives in computer systems. Some of the advantages of SSDs include that they: • Are much faster than hard drives.

	• Have no moving parts, so they last longer. • Have lower power consumption than hard drives. • Are less susceptible to physical damage. • Are smaller and lighter than hard drives. • Use the same SATA interface used by standard hard disk drives. The main disadvantage currently for solid state drives is cost--they are several times more expensive than comparable hard drives.
External flash storage devices	Like an SSD, external flash storage devices store information using programmable, non-volatile flash memory. External flash storage devices most commonly connect to the computer using a USB interface. Advantages of flash devices include: • Portability • Larger storage capacity than optical discs • Relatively fast read access. Some of the disadvantages of flash devices are: • Less storage capacity than hard disks • Relatively slow write speeds Common external flash storage devices include: • CompactFlash cards • eMMC cards • SD cards • SSD cards • MiniSD cards • MicroSD cards • xD cards • Hybrid cards (combines SSD and HDD technology) • Memory sticks
Optical discs	Optical discs, such as CDs, DVDs, and Blu-ray discs, store information using pits and lands in the surface of their reflective coating. As the disc spins, the optical drive uses a laser to read data stored on the surface of the disc in the form of deflected and reflected light. Some advantages of optical discs include the following: • They are highly portable. • They are inexpensive. • Recordable optical discs allow you to both read and write data. • They have a long shelf life and are relatively sturdy. • They use the same storage interfaces used by standard hard disk drives (SATA, SCSI, and PATA). • Blu-ray discs can store a large amount of data (25 GB or more, depending upon the format). Some disadvantages of optical discs include: • They are slower than hard disks, SSDs, and external flash devices. • Older optical disc standards have limited storage capacities (650 MB for CDs, 4.7 GB for DVDs). • There are occasionally compatibility issues between disc formats and readers.

Storage devices in Linux are represented by device files.

- Device files are located in the **/dev** directory.
- The **/dev** directory contains files for all types of devices, even those that don't exist on the system.

- Not only do device files represent devices, they also indicate how data is transferred to that device:
- Devices, such as hard drives, that receive data in block transfers by using memory to buffer the transfers are called *block* devices.
- Devices that send data transfers character-by-character (like a keyboard) are called *character* devices.

  The table below lists and describes the most common device files:

Device File	**Description**
/dev/sd*xn*	*sd* files identify hard drives. A letter (beginning with *a*) follows the *sd* designation and identifies the ID of the hard drive. At the end is appended a number (beginning with 1) that identifies the partition on the drive. Examples include: • sda2 is the second partition (2) on the hard drive with the lowest ID number (a). • sdc1 is the first partition (1) on the hard drive with the third lowest ID number (c). • sda1 is the first partition (1) on the hard drive with the lowest ID number (a). • sdb3 is the third partition (3) on the hard drive with the second lowest ID number (b). • sdc2 is the second partition (2) on the hard drive with the third lowest ID number (c). • sdd1 is the first partition (1) on the drive with the forth lowest ID number (d).
/dev/sr*n*	This is a special designation used to identify optical drives in the system. The optical drive with the lowest ID number is addressed as **sr0**, the optical drive with the next lowest ID number is addressed as **sr1**, and so on. Many distributions include symbolic links named **/dev/cdrom** or **/dev/dvd** that point to the actual device file (**sr0**).
/dev/fd*n*	*fd* files identify floppy drives. Device numbering begins at 0. For example, **/dev/fd0** is the first floppy drive.
/dev/tty*n*	*tty* files identify local terminals on the system. Device numbering begins at 0. Subsequent terminals are represented with files that increment by one (e.g., the file for terminal two is /dev/tty2, and so on).
/dev/ttyS*n*	*ttyS* files identify serial ports. Device numbering begins at 0. Files for subsequent serial ports are represented by files that increment by one (e.g., the file for serial port two is **/dev/ttyS1**, and so on).
/dev/lp*n*	*lp* files identify parallel ports. Device numbering begins at 0. Files for subsequent parallel ports are represented by files that increment by one (e.g., the file for parallel port two is **/dev/lp1**, and so on).
/dev/st*n*	st files identify SCSI tape devices. Device numbering begins at 0.

From <https://cdn.testout.com/client-v5-1-10-414/startlabsim.html>

7.2.3

Monday, April 24, 2017 10:44 AM

The *Globally Unique Identifier Partition Table* (GPT) scheme has been introduced as a replacement for the *Master Boot Record* (MBR) partitioning scheme. GPT has several advantages over using MBR. GPT:

- Uses only one type of partition. There are no primary, extended, or logical partitions.
- Supports extremely large storage devices and partitions.
- Allows up to 128 partitions on a storage device.
- Stores a copy of the partition table in the first and last sectors of the storage device. If one copy gets corrupted, then the redundant copy can be used instead.
- Verifies the integrity of the partition table using a *cyclic redundancy check* (CRC).
- Assigns unique IDs to each storage device and partition.

The following commands can be used to manage GPT partitions:

Use	To	Examples
gdisk	Do the following: • Create and delete GPT partitions. • Display information about a partition. • Change the name and type of a partition. • Verify a hard disk. • Back up and restore a disk's partition table. • Convert an MBR partition table to a GPT partition table. The syntax for using gdisk is **gdisk *device_name***. The following options can be used within gdisk: • **?** displays the help screen. • **b** backs up GPT information to a file. • **c** changes a partition's name. • **d** deletes a partition. • **i** displays detailed partition information. • **l** lists partition type codes. • **n** adds a new partition. • **o** creates a new GUID partition table. • **p** prints the partition table. • **q** quits gdisk without saving changes. • **s** sorts the list of partitions. • **t** changes a partition's type code. • **v** verifies a storage device. • **w** writes changes to the partition table of the storage device and exits gdisk.	**gdisk /dev/sdc** opens gdisk and edits the partition table on the third storage device in the system.
parted	Do the following: • Create and delete GPT partitions. • Modify GPT partitions. The **parted** command writes partition changes to disk immediately. Carefully plan any partition changes to be made before using **parted**.	**parted** starts the parted utility.

	The syntax is to run **parted** at the shell prompt. The following commands can be used within **parted**: • **select** *device_name* identifies which storage device to edit. • **mkpart** ***partition_type start_point end_point*** creates a new partition. For example: • To create a standard Linux partition, specify a partition type of **Linux**. • To create a partition that starts at 1GB and ends at 21 GB, specify a start point of **1024** and an end point of **21504**. • **print** displays a list of partitions on the device. • **name** ***partition_name*** renames a partition. • **move** ***partition start_point end_point*** moves a partition to a different location on the storage device. • **resize** ***partition start_point end_point*** resizes a partition. • **rm** ***partition*** deletes a partition.	

From <https://cdn.testout.com/client-v5-1-10-414/startlabsim.html>

7.3.3

Monday, April 24, 2017 10:45 AM

The Logical Volume Manager (LVM) provides an alternative method to managing partitions on a Linux system. LVM gives a system administrator more flexibility in allocating storage on a system. Important aspects of LVM include:

- You can (within certain limits) resize and move logical volumes while they are still mounted and running.
- Logical volumes may be identified by using descriptive names (e.g., research or marketing) instead of physical disk names such as sda and sdb.

The following table describes common LVM commands:

Command	Description	Examples
pvcreate	Initializes physical volume for later use by the LVM.	**pvcreate /dev/sdb** creates a physical volume from the second hard disk in the system. **pvcreate /dev/sdd1** creates a physical volume from the first partition on the fourth hard disk in the system.
pvscan	Scans all disks for physical volumes and displays all found physical volumes on the system and their associated volume groups.	
vgcreate	Creates a new volume group.	**vgcreate system /dev/sdb** creates a volume group named system and adds the /dev/sdb physical volume to the group. **vgcreate backup /dev/sdd** creates a volume group named backup and adds the /dev/sdd physical volume to the group.
vgextend	Adds one or more physical volumes to an existing volume group, increasing its available storage space.	**vgextend system /dev/sdc** adds the /dev/sdc physical volume to the existing system volume group.
lvcreate	Creates a new logical volume from the space available in a volume group. Options include: • **-L** specifies the size. Use the following size suffixes: • **K** for kilobytes • **M** for megabytes • **G** for gigabytes • **T** for terabytes • **P** for petabytes • **E** for exabytes • **-n** specifies the name.	**lvcreate -L 20G -n data system** creates a 20 GB logical volume named data in the system volume group. **lvcreate -L 2T -n Storage1 backup** creates a 2 TB logical volume named Storage1 in the backup volume group. **lvcreate -L 1T -n Storage2 backup** creates a 1 TB logical volume named Storage2 in the backup volume group.

lvscan	Scans all known volume groups in the system for logical volumes and displays the result.	
lvextend	Extends the size of a logical volume. The **-L** option is used to specify the new size of the volume.	**lvextend -L 30G data** extends the data logical volume to a total of 30 Gigabytes. **lvextend -L +10G data** extends the data logical volume by another 10 Gigabytes.

You can also use the following commands to manage an LVM-based storage configuration:

- Use **pvmove** to move the data from one physical volume to another physical volume.
- Use **pvremove** to remove the LVM label from a device so that it will no longer be recognized as a physical volume.
- Use **vgreduce** to remove a physical volume from an existing volume group. Before running vgreduce, you must use pvmove to shift the data to another physical volume.
- Use **vgremove** to delete a volume group. Before you can use vgremove, you must first remove all logical volumes that have been defined in the volume group.
- Use **lvreduce** to reduce the size of a logical volume.
- Use **lvremove** to remove a logical volume from the system.

After your logical volumes have been created, you need to create file systems on them and then mount them:

- You create a file system using mkfs, just as with traditional partitions. Use the following syntax: **mkfs -t *file_system* /dev/*volume_group*/*logical_volume***
- You mount a logical volume using the mount command, just as you would to mount file systems on traditional partitions. Use the following syntax: **mount -t *file_system* /dev/*volume_group*/*logical_volume* /*mount_point***

From <https://cdn.testout.com/client-v5-1-10-414/startlabsim.html>

7.4.5

Monday, April 24, 2017 10:45 AM

The file system determines how a computer's files are organized on a hard drive. Linux supports many different file system types. The following table describes several common file systems:

Type	Description
ext2	The Second Extended File System (ext2) is one of the oldest Linux file systems still available. • ext2 stores data in a standard directory and file hierarchy. • The maximum file size supported is 2 TB. • An ext2 volume can be up to 4 TB in size. • File names can be up to 255 characters long. • Linux users, groups, and permissions are supported. • It does not use journaling (which is used in most modern file systems). As a result, ext2 takes a long time to recover if the system shuts down abruptly.
ext3	The Third Extended File System (ext3) is an updated version of ext2 that supports journaling. Before committing a transaction to a storage device, the ext3 file system records the transaction to the journal and marks it as incomplete. After the disk transaction is complete, the file system marks the transaction as complete in the journal. By doing this, ext3 can keep track of the most recent file transactions and whether or not they were completed. This allows ext3 to recover much more quickly than ext2 in the event of an unclean system shutdown.
ReiserFS	The Reiser file system (ReiserFS) is an alternative to the ext3 file system. Like ext3, Reiser uses journaling to make crash recovery very fast. However, Reiser is a completely different file system from ext2 and ext3, using a dramatically different internal structure. ReiserFS supports a maximum file size of 8 TB and maximum volume size of 16 TB. In addition, the structure of Reiser allows it to perform much faster than ext2 or ext3.
ext4	ext4 is the fourth generation file system in the ext file system family. ext4 includes all of the features found with ext2 and ext3, with the addition of the following features: • Support for file sizes up to 16 TB and disk sizes up to 1 exabyte (EB) • Allows for up to four billion files in the file system • Uses checksums to verify the integrity of the journal file itself Checksums help improve the overall reliability of the system because the journal file is the most heavily used file of the disk.
swap	A swap file system is used as virtual memory (the portion of the hard disk used to temporarily store portions of main memory) by the operating system. A recommended practice is to make the swap file size between 1 and 1.5 times the amount of memory on the computer.
NTFS	Microsoft operating systems use NTFS (New Technology File System). Linux provides limited support for NTFS.
VFAT	VFAT is a FAT32 file system for Linux and does not support journaling. VFAT includes

	long name support. Support for VFAT must be compiled into the kernel for the system to recognize the VFAT format.
XFS	The XFS file system was developed for the Silicon Graphics IRIX operating system. An XFS file system is proficient at handling large files, offers smooth data transfers, and provides journaling. It also can reside on a regular disk partition or on a logical volume.
Btrfs	Btrfs is a Linux file system that uses a *copy-on-write* file system. Using copy-on-write technology, Btrfs provides several key features not found in earlier file systems: • **Storage pools** - Instead of using traditional disk partitions, Btrfs allows you to create *storage pools* from the storage devices in your system. From the storage pool, you can then allocate space to specific *storage volumes*. Instead of mounting partitions, you mount storage volumes at mount points in the file system. • **Snapshots** - The snapshot functionality provided by Btrfs protects data. It can be configured to take snapshots of your data at specified intervals and save it on separate media. If a file ever gets lost or corrupted, you can restore a previous version of the file from a snapshot.

A disk partition must be formatted using a file system. The following table describes the commands needed to format a partition:

Command	Description	Example
mkfs	Creates an ext family file system or a fat file system. The **mkfs**command uses the following options: • **-t *[file_system_type]*** determines the file system. File system types include: • **ext2** (identical to the **mkfs.ext2** command) • **ext3** (identical to **mkfs.ext3**) • **ext4** (identical to **mkfs.ext4**) • **vfat** (identical to **mkfs.vfat**) • **reiserfs** (identical to **mkreiserfs**) • **-b** specifies the block size. Supported values are 1024, 2048, or 4096. • **-i** determines how many inodes are on the partition and uses the same values as **-b**. • **-j** appends a journal to an ext2 file system. Without the **-b** and **-i** options, **mkfs** calculates the optimal values for you automatically.	• **mkfs -t ext2 /dev/sda4** creates an ext2 file system on the fourth partition on the first hard disk drive. • **mkfs -t ext3 /dev/sda1** creates an ext3 file system on the first partition on the first hard disk drive. • **mkfs -t ext3 /dev/sdc2** creates an ext3 file system on the second partition on the third hard disk drive. • **mkfs -t ext4 /dev/sdb1** creates an ext4 file system on the first partition on the second hard disk drive.
mkreiserfs	Creates a ReiserFS.	• **mkreiserfs /dev/sda2** formats the second partition on the first hard disk with the Reiser file system.
mkswap	Creates a swap partition. A swap partition is the location on the hard drive where an operating system writes memory information when it runs out of RAM. • The **swapon** command must be run to activate the swap partition.	• **mkswap /dev/sda2** formats the second hard drive as the swap partition. • **swapon /dev/sda2** activates second hard drive as the swap partition.

	• The **swapoff** command is used to deactivate swap partitions. Both **swapon** and **swapoff** use the **-a** option to enable or disable all swap partitions listed in **/etc/fstab**.	• **swapon -a** activates all swap partitions. • **swapoff /dev/sda2** deactivates second hard drive as the swap partition. • **swapoff -a** deactivates all swap partitions.
mke2fs	Create an ext2, ext3, or ext4 file system. Command options include: • **-b** specifies the block size of the file system in bytes (valid sizes are 1024, 2048, and 4096 bytes per block). • **-j** creates the file system with an ext3 journal. • **-L** sets the volume label for the file system. • **-n** displays what **mke2fs** would do if it created a file system, but does not actually create the file system. • **-t** specifies the file system type (e.g., ext2, ext3, ext4, etc.) to be created.	• **mke2fs /dev/sda2** creates an ext2 file system on the second partition on the first hard disk drive. • **mke2fs -j /dev/sda1** creates an ext3 file system on the first partition on the first hard disk drive. • **mke2fs -t ext4 /dev/sdc3** creates an ext4 file system on the third partition on the third hard disk drive.

Keep the following in mind when working with file systems:

- You cannot format an extended partition. However, you can create logical partitions inside an extended partition and format them.
- File systems use an inode (information node) table to store information about files. An inode specifies where a file's data physically exists on a disk. Inodes also contain additional information including:
- File size
- Modification, access, and creation times
- Permissions
- Ownership
- Each file system has a *superblock*, which contains information about the file system, such as:
- File system type (e.g., ext2 and ext3)
- Size (e.g., 10GB and 360GB)
- Status
- Linux maintains multiple redundant copies of the superblock in every file system.

From <https://cdn.testout.com/client-v5-1-10-414/startlabsim.html>

7.5.3

Monday, April 24, 2017 10:47 AM

Mounting is the process of making a storage device accessible to users through the directory tree. The directory in which the device is mounted is called the *mount point*.

- Partitions and LVM logical volumes are represented by device files located in the **/dev** directory. However, these storage devices must be mounted before the data on them can be accessed.
- A storage device can be mounted in a directory in the file system. When accessing the directory in the file system, you are actually accessing the device mounted in that directory.
- You should mount storage devices in empty directories. Mounting a volume to a directory that contains data makes the data inaccessible.
- The **/mnt** and **/media** directories (depending on the system configuration) are directories that contain mount points specifically for external storage devices (e.g., CD-ROM drives, floppy drives, magnetic tape drives).

The following files manage and monitor the file system mounting:

File	Description
/etc/fstab	The **/etc/fstab** file identifies devices to mount each time the system boots. When the system boots, it automatically mounts the volumes identified in the file. The file contains entries with six fields that control how a device is mounted. The following is a typical fstab entry: `/dev/sda3 /mnt/disk1 ext3 auto,ro,nosuid,users 0 1` An entry consists of the following variables, which are described below: [*device_to_mount*] [*mount_point*] [*file_system_type*] [*options*] [*dump*] [*fsck*] • *Device_to_mount* is the path to the device file or the label that describes the storage device to be mounted. • *Mount_point* specifies where to mount the device. This is the directory where the data on the device can be accessed. • *File_system_type* specifies the type of file system that has been created on the storage device. • *Options* specify the additional options to be used when mounting the device. Multiple options are separated by commas. • **sync** enables synchronous I/O. Changes are written to disk immediately. Usually used for removable storage devices. (**async** disables this function.) • **async** enables asynchronous I/O. Changes are cached and then written when the device isn't busy. Usually used for non-removable devices such as hard drives. (**sync** disables this function.) • **atime** updates the timestamp on each file's inode. (**noatime** disables this function.) • **auto** allows the device to be mounted automatically when the system boots. • **noauto** prevents the device from being mounted automatically when the system boots. • **dev** allows block files to be read from the device. (**nodev** disables this function.) • **exec** allows programs and script files in the file system to be run. (**noexec** disables this function.) • **owner** identifies that only the device owner can mount the file system. • **ro** mounts the storage device as read-only. • **rw** mounts the storage device as read/write.

	• **suid** allows the SUID bit to be set on files in the file system. (**nosuid** disables this function.) • **nouser** allows only the root user to mount the file system. • **users** allows any user to mount the file system. • **defaults** uses the following default settings: rw, suid, dev, exec, auto, nouser, and async. • *Dump* determines whether the file system needs to be dumped. If set to a value of **0**, it is assumed that the file system does not need to be dumped. If set to a value of **1**, the file system will be dumped. • *fsck* determines the order in which to run **fsck** (file system check) during system boot. This field should always be set to a value of **1** for the device containing the root file system (/). All other file systems should be set to a value of **2**.
/etc/mtab	The **/etc/mtab** file tracks the currently mounted volumes on the system.
/procs/mounts	The **/procs/mounts** file contains entries for all currently mounted volumes on the system.

Use the following commands to manage the file system mounting:

Command	Description	Example
mount /dev/[*device*] [*mountpoint*]	Mount a volume or device. Common mount options: • **-a** mounts all file systems listed in the **/etc/fstab** file. • **-r, ro** mounts the volume as read only. • **-w, rw** mounts the volume as read/write. • **-t** specifies the volume type (If you mount an ext3 file system without the **-t**, the system recognizes it as an ext2 file system). • **-o loop** mounts an ISO file.	• **mount -a** reads the **/etc/fstab** file and mounts all volumes listed (except those with the **noauto** option). • **mount -rt reiserfs /dev/sdc1 /mnt/reis** mounts the sdc1 device using the Reiser file system read-only to the **/mnt/reis** mount point. • **mount -t iso9660 /dev/sr0 /media/cdrom** mounts an optical disc device to the**/media/cdrom** mount point. • **mount -wt vfat /dev/fd0 /mnt/floppy**mounts the fd0 device with the VFAT file system as read/write to the floppy mount point.
mount	View the currently mounted volumes on the system.	• **mount /etc/mtab** displays the contents of the **/etc/mtab** file.
df	View which file systems are mounted to specific mount points.	
umount [*device*] umount[*mountpoint*]	Unmount a volume or device from the system. If a "disk is busy" error message is displayed when unmounting a device: • Make sure the current working directory is not in that file system. • Close any open files located on that file system.	• **umount /dev/sdc1** unmounts the sdc1 device. • **umount /mnt/reis** unmounts the device on the /mnt/reis mount point. • **umount /dev/sr0** unmounts the optical disc device. • **umount /mnt/cdrom** unmounts the device on the /mnt/cdrom mount point (most likely an optical disc).

From <https://cdn.testout.com/client-v5-1-10-414/startlabsim.html>

7.6.3

Monday, April 24, 2017 10:48 AM

Use the following commands to maintain file system integrity:

Command	Description	Examples
df	Displays the free space in the partition holding the specified directory. If no directory is given, the space available on all currently mounted file systems is shown. Disk space is shown in 1 K blocks by default. Common options include: • **-h** displays the output in get human readable format (bytes, KB, MB, GB, TB). • **-i** displays inode information. • **-l** limits the list to local file systems.	**df /home** lists the free space on the partition that holds the **/home** directory.
du	Displays files and file sizes in and below a specified directory. Common options include: • **-c** lists a total amount of space used in the directory. • **-h** display the output in human readable format (bytes, KB, MB, GB, TB). • **-s** lists only the total, not each file. • **-a** evaluates all files, not just directories.	**du -c /home/badam** lists all files and directories in badam's home directory along with a file size and a total amount of space taken up by the directory. **du -c -s /home/badam** shows the total amount of space taken up in badam's home directory.
lsof	Displays open files in the file system. **lsof** gives the following information by default: • The command used to access the file • Process ID • Name of the user who is accessing the file • A file descriptor (these are described in the **lsof** man pages) • File node type • Device numbers • File size • Inode address • File path Common options include: • **+D [*directory_name*]** recursively lists files in a directory. • **-c [*command_name*]** lists all files for processes that are executing the specified command. • **-u [*user*]** lists open files owned by the specified user. • **-g [*process_ID*]** lists files opened by a specific process.	**lsof -u *user*** lists files opened by processes that the specified user owns.

fuser	Displays the process IDs of processes that are accessing a specified file or file system. Common options include: • **-a** displays all process IDs. • **-v** displays extended information (verbose mode). • **-u** appends the username to each process ID. • **-m** displays process IDs in a specified directory.	**fuser -v /bin/bash** shows the user name and process ID for all users who have an open bash shell.
fsck	Checks and optionally repairs one or more Linux file systems. Common options include: • **-s** serializes **fsck** when multiple file systems are checked. • **-t** specifies the type(s) of file system to be checked. • **-a** automatically repairs the file system without any questions. • **-r** prompts for confirmation when errors are found and ask permission to fix the errors (only when **-a** is not specified). Be aware of the following: • The file system must be unmounted before using **fsck**. • When manually running **fsck**, use runlevel 1 (init) or rescue.target (systemd) to ensure that other users do not mount the file system.	**fsck -t ext3 /dev/sdb1** checks the first partition on the first partition of the second hard drive.
e2fsck	Checks and optionally repairs ext2, ext3, or ext4 file systems. Common options include: • **-f** forces a file system check even when the file system appears clean. • **-n** opens the file system as read-only and automatically answers all questions as "no". • **-p** automatically repairs the file system without any questions. • **-y** automatically answers all questions as "yes". • **-b** uses an alternative superblock if the primary superblock is corrupt.	**e2fsck -p /dev/sda1** checks and repairs the first partition of the first hard drive so long as it is partitioned using ext2.
debugfs	Debugs the file system. The command examines and changes the state of an ext2, ext3, or ext4 file system. It allows administrators to unlink directories, change inode blocks find all inodes that point to a block, and several other similar functions. Command options include: • **-w** specifies the file system should open in read-write mode. • **-c** specifies that the file system should open in catastrophic mode (this is useful for file system with significant corruption). • **-f** executes commands in a text file.	**debugfs /dev/sda1** opens the first partition of the first hard drive and displays a prompt that administrators can use to execute commands for the file system.
dumpe2fs	Prints super block and block information for an ext2, ext3, or ext4 file system. This includes	**dumpe2fs /dev/sda1** lists information for the first

	information for each sector on the partition about sector type, block ranges, inode information, free blocks, and similar information. Command options include: • **-b** prints blocks reserved as bad in the file system. • **-h** prints only super block information. • **-x** prints group information block numbers in hexadecimal format.	partition of the first hard drive.
tune2fs	Adjusts tunable file system parameters on ext2, ext3, and ext4 file systems. Some of the adjustable parameters include volume label, reserved blocks, inode sizes, and journaling. Tune2fs can also implement access control lists for individual users. Command options include: • **-c** adjust the number of mounts after which the file system will be checked. • **-e remount-ro** remounts the file system as read-only. • **-l** lists the contents of the file system super block. • **-o acl** enables Posix access control lists. • **-j** converts ext2 file systems to ext3 file systems.	**tune2fs -o acl /dev/sdb1** enables access control lists on the first partition of the second hard drive. The drive needs to be remounted.
xfs_info	Displays the XFS file system parameters, such as the block size and inode data structures. This is the same functionality as the **xfs_growfs -n** command.	**xfs_info /dev/sdb1** displays file system parameters for the first partition of the second hard drive.
xfs_metadump	Copies (dumps) the metadata from an XFS file system to a file. It does not alter the file system. By default, the file names and extended attribute names are obfuscated before they are dumped. Command options include: • **-e** stops dumping the file system if there is a read error. • **-g** displays the dump process. • **-o** disables obfuscation of file names and extended attributes. Only use **xfs_metadump** to dump unmounted, read-only mounted, or frozen file systems.	**xfs_metadump -o /dev/sdb1 /dump** copies the file system metadata for the first partition on the second hard drive to the **/dump** file.

From <https://cdn.testout.com/client-v5-1-10-414/startlabsim.html>

7.7.3

Monday, April 24, 2017 10:48 AM

Disk quotas prevent a user or group from using a disproportionate amount of disk space on a volume. Quotas can be configured to:

- Limit the number of files and directories a user or group can create.
- Limit the amount of disk space a user or group can use.

Quota types include the following:

- A soft limit allows the user to temporarily exceed a quota limit.
- A hard limit is a fixed limit that the user cannot exceed.

The following table lists the general steps to implement quotas:

Step	Procedure
Install quota package	Use yum, zypper, or apt-get to install the **quota** package on the system where quota limits will be set.
Edit mount options in **/etc/fstab**	Edit the **/etc/fstab** file to add the mount options for the file system to enable quotas: • **usrquota** enables quotas for users. • **grpquota** enables quotas for groups.
Create quota files	Create the **aquota.user** and **aquota.group** files in the directory where the partition is mounted.
Enable quotas and view a quota report	Enable disk quotas and then generate a disk usage and quota report. The report shows: • How much space to allocate to each user. • How much space is currently consumed by each user. • Whether some users are using a significant amount of disk space.
Edit quotas	Edit a quota for the specified user or group. Be aware of the following when editing quotas: • Set the soft and hard quotas for blocks. This limits the total amount of disk space per user or group. • Set the hard and soft quotas for inodes. This limits the total number of files and directories per user or group. • Users may exceed soft quotas for a number of days specified in the grace period (seven by default). When the grace period expires, users cannot create additional files. • Users cannot exceed hard quotas. • When setting block quotas, 1000 blocks is about 1 MB, and 1,000,000 blocks is about 1 GB. • Setting the quota limits to 0 removes all quotas.

The table below describes common commands for working with quotas:

Use	To	Examples
quotacheck -mavug	To create the **aquota.user** and **aquota.group** files in the file system (after placing the quota entries in **/etc/fstab**). Common options include: • **-m** updates the quota database even if other	**quotacheck -mavug /home** creates the **aquota.user** and **aquota.group**files in the root (/) directory.

	processes are running on the file system. • **-a** updates the quota database. • **-v** runs the command in verbose mode. • **-u** and **-g** run the database updates for users and groups, respectively.	
quotaon	Enable quotas for the mounted file system: • **-a** enables all mounted file systems listed in **/etc/mtab**. • **-v** runs the command in verbose mode.	**quotaon -av /** enables quotas for the root (/) directory.
quotaoff	Disable quotas for the mounted file system.	**quotaoff /home** disables quotas for the **/home** directory.
repquota	Display a summary of the disc usage and quotas for the specified file systems, including the specific number of files and used space by user. Common options include: • **-v** reports all quotas, even if there is no usage. • **-n** does not resolve user and group names to speed printing time. • **-u** and **-g** report for users and groups, respectively. • **-a** gives information for all file systems listed in **/etc/mtab**.	**repquota /home -uv** creates a user quota report for the **/home** directory.
edquota	Open and edit a user's quota, a group's quota, or change the grace period: • **-u** changes the user's quota. • **-g** changes a group's quota. • **-t** changes the grace period.	**edquota -u mtomm** opens the quota file for the mtomm user account.
quota	Display the current user's quota: • **-u** shows the quota for a user. • **-g** shows the quota for a group. • **-v** shows current the current usage, the hard quota and the soft quota for blocks and inodes.	**quota** displays the quota report only for the current user account. **quota -u dhanson** displays the quota report only for the dhanson user account.

From <https://cdn.testout.com/client-v5-1-10-414/startlabsim.html>

7.8.3

Monday, April 24, 2017 10:48 AM

When a user creates a file (or directory), the user and the user's primary group receive ownership for the file (or directory) by default. To change the user that owns a file, you must be logged in as root. To change the group that owns a file, you must be logged in as root or as the user who currently owns the file.
The table below lists the most common commands for managing file ownership:

Use	To	Example
ls -l	View a long listing of files and directories. The long listing shows the mode of each file and directory along with ownership information.	**drwxr-xr-x 22 root root 4096 Jun 19 15:01 sales** (Root is the file owner and the group owner in this example.)
chown	Change the ownership of a file or directory. Be aware of the following options: • **-R** changes the ownership of the file recursively throughout the directory tree. • ***user*** changes the file ownership only. • ***user*:*group*** or ***user*.*group*** change the user and group ownership of the file. • **:*group*** or **.group** changes the group ownership only.	**chown pmorril /sales/report** makes pmorril the user owner of the /sales/report file. **chown -R pmorril /sales** makes pmorril the owner of all files in the /sales directory (and below). **chown pmaxwell:sales /sales/report** makes pmaxwell the user owner and sales the group owner of the file. **chown :sales -R /sales** makes the sales group the owner of all files in the /sales directory.
chgrp	Change the group owner of a file or directory.	**chgrp sales /sales/report** makes the sales group the group owner of the file.

From <https://cdn.testout.com/client-v5-1-10-414/startlabsim.html>

7.9.4

Monday, April 24, 2017 10:50 AM

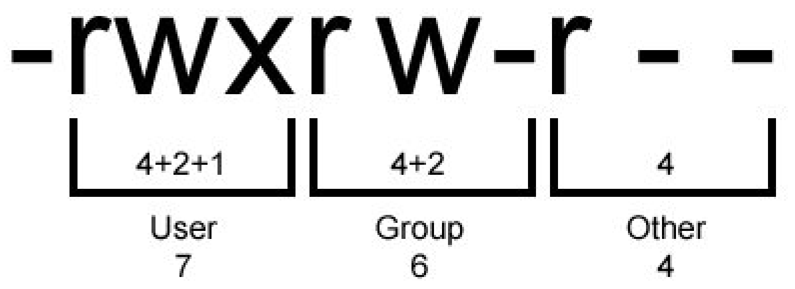
Every file and directory in the Linux file system has an *inode* (information node) that stores information about the file or directory, including when it was last modified, size, data block location, permissions, and ownership. The portion of the inode that stores permission information is called the *mode*. The mode has three sections:

- User permissions (owner)
- Group permissions (group owner)
- Other permissions (everyone else on the Linux system who is not an owner or a member of the owning group)

There are three types of permissions contained in the mode, each of which is described in the table below:

Permission	**Letter Abbreviation**	**Octal Value**	**Allowed Actions on Files**	**Allowed Actions on Directories**
Read	r	4	Open and read the file	List directory contents if the execute permission is also present
Write	w	2	Edit the file and save the changes	Add, delete, and rename files if the execute permission is also present
Execute	x	1	Execute the file, if it's a program file or a shell script (must be used in conjunction with the read permission)	Enter the directory and access its contents

Permissions are identified with either the letter abbreviation (**r**, **w**, or **x**), or the octal value that corresponds to the permission. The following graphic depicts how permissions are referenced:



In the mode:

- A **d** preceding the permissions indicates that the object is a directory.
- A dash (-) preceding the permissions identifies a file (the example above is for a file).
- Permissions are grouped according to user, group, or other permissions.
- If a given permission has *not* been assigned, a dash (-) takes its place in the mode.
- When using numbers to represent permissions, add the numbers together within each permission group. Then string the numbers together. For example, the mode of the file in the graphic above can be represented by the number 764.
- The root user has all permissions to files and directories regardless of the mode settings.

The table below lists the most common commands for managing permissions:

Use	To	Example
ls -l	View a long listing of files and directories. A long listing displays the permissions assigned to files and directories (among other information)	**drwxr-xr-x** 22 root root 4096 Jun 19 15:01 sales (This is a directory with 755 permissions assigned.)
chmod	Change the permissions for the specified file. You can use the following syntax options: • ***entity+permission*** adds a permission for a user, group, or other to a file or directory. • ***entity-permission*** removes a permission for a user, group, or other from a file or directory. • ***entity=permission*** sets the permission equal to the permission specified for a user, group, or other for a file or directory. • **decimal_value** sets the permissions for the file according to the numbers represented for each mode entity. • **-R** sets permissions recursively.	**chmod u+x,g+x,o+x myfile** adds the execute permission to the myfile file for user, group, and other. **chmod g-w,o-w myfile** removes the write permission for group and other from the myfile file. **chmod u=rwx myfile** grants the user read, write, and execute permissions for the myfile file. **chmod 711 myfile** grants the user read, write, and execute permissions (7) while group and other both receive execute permission (1) for the myfile file.

From <https://cdn.testout.com/client-v5-1-10-414/startlabsim.html>

7.10.3

Monday, April 24, 2017 10:50 AM

The umask command changes the default file and directory permissions. By default, files receive rw-rw-rw- (666) permissions and directories receive rwxrwxrwx (777) permissions when they are created. In most cases, the default assignment gives excess permissions to files and directories.
The umask value identifies which permissions are *removed* from the default permissions when files and directories are created. The following table shows what happens when the mask is set to a value of 022.

Umask Calculation	Files (binary)	Directories (binary)	Files (letter abbreviation)	Directories (letter abbreviation)
Default Permission	666	777	rw-rw-rw-	rwxrwxrwx
Umask (minus)	022	022	----w--w-	----w--w-
Result (equals)	644	755	rw-r--r--	rwxr-xr-x

Additional example umask calculations are described below:

- A umask of **066** results in file permissions of **rw------- (600)** and directory permissions of **rwx--x--x (711)**.
- A umask of **033** results in file permissions of **rw-r--r-- (644)** and directory permissions of **rwxr--r-- (744)**.
- A umask of **011** results in no changes to file permissions (the **x** permission is already removed by default) and directory permissions of **rwxrw-rw- (766)**.

The table below lists the commands for managing umask:

Command	Description	Example
umask	Displays the current umask setting	022 is the typical default umask setting.
umask*number*	Changes the default umask.	Typing **umask 007** sets the umask to remove nothing from the user or group, but removes all permissions from other.

Be aware of the following:

- The default umask value may vary depending on the Linux distribution (022 or 0022 is the most common default).
- Setting the umask with the **umask** command is only persistent for the shell session.
- To make the umask persistent through shell sessions and reboots, add the **umask** command to the shell configuration file (depending on the distribution).

From <https://cdn.testout.com/client-v5-1-10-414/startlabsim.html>

7.11.3

Monday, April 24, 2017 10:50 AM

Be aware of the following special permissions:

Permission	Letter Abbreviation	Example	Octal Value	Description
SUID (Set User ID)	**s** in the execute permission position of the user permissions	rw**s**rw-rw-	4	If the SUID bit is set, the program will run with the permissions of the file owner, not with the permissions of the user who runs the program. • The most common use of SUID is to allow users to run a command as the root user. • Users do not become the root user, but rather the command or program runs as if executed by the root user. • Some programs require the SUID bit set for proper functionality. • Be careful in setting the SUID bit as it could give a program too many permissions.
SGID (Set Group ID)	**s** in the execute permission position of the group permissions	rwxrw**s**rw-	2	If the SGID bit is set: • On a file, the program will run with the group permissions of the group owner. • On a directory, a newly created file will receive the same group owner as assigned to the parent directory.
Sticky bit	**t** in the execute permission position of the other permissions	rwxrw-rw**t**	1	This marks the file (not directory) in such a way as to prevent the file's deletion from the system by anyone except the file owner. Setting the sticky bit works particularly well with shared files.

Use the following commands when managing special permissions:

Command	Description	Example
ls -l	Displays a long file listing. A long file listing shows the permissions for the files (among other information).	**drwsr-xr-x** 22 root root 4096 Jun 19 15:01 sales (This is a script with 4755 as its mode and has the SUID set.)
chmod	Assigns a special permission. Be aware of the following syntax options: • ***[decimal_value]*** sets the permissions for the file according to the numbers represented for each mode category. • The special permission *precedes* the standard octal representation of a set of permissions.	**chmod 4*xxx*** sets the SUID. **chmod u+s** sets the SUID. **chmod u-s** removes the SUID. **chmod 2*xxx*** sets the SGID. **chmod g+s** sets the SGID. **chmod 1*xxx*** sets the sticky

	• Only the *first* number changes to identify the special permission group settings. • ***[category]***+***[permission]*** adds a special permission for a user, group, or other (category) to a file. • ***[category]***-***[permission]*** removes a special permission for a user, group, or other from a file.	bit. **chmod u+t** sets the sticky bit. **chmod u-t** removes the sticky bit. **chmod 6*xxx*** sets both the SUID and SGID. **chmod 7*xxx*** sets the SUID, GUID, and sticky bit.

From <https://cdn.testout.com/client-v5-1-10-414/startlabsim.html>

7.12.5

Monday, April 24, 2017 10:51 AM

The **tar** (tape archive) utility takes the contents of several files and stores them as a single file. The tar command can be used to backup directories or entire file systems. By convention, an archive file created with tar is assigned an extension of **.tar** to help others know that the tar utility must be used to extract files from it.
The following table lists several ways that tar can be used to create and compress archive files:

Use	To	Examples
tar	Combine multiple files into a single file. Options include: • **-A** appends one tar file to another archive file. • **-c** creates a new archive. • **-d** identifies differences between the files in an archive file and the same files in the file system. • **-v** displays a list of all files being written into the archive. • **-f** specifies the file to create or unpack. Without this option, tar uses standard input and output as the destination. • **-x** extracts the files. If no destination directory is specified, then tar extracts the files to the current working directory. • **-z** compresses and decompresses a file using the **gzip** utility (normally named with a .gz extension). • **-j** compresses and decompresses a file using the **bzip2** utility (normally named with a .bz2 extension). • **-J** compresses and decompresses a file using the **xz** utility (normally named with a .xz or .lzma extension). • **-C** changes to a specific directory to extract the files. • **-t** lists the contents of an archive. • **-P** tells tar to not strip the leading / from filenames as they are added to the archive. • **-r** adds files to the end of an existing tar archive. • **-u** adds files to the end of an existing tar archive only if they are	**tar -cf /root/tarbackups/oct17backup.tar /home** writes a backup of the /home directory to the**/root/tarbackups/oct17backup.tar** file. **tar -cvf /root/tarbackups/oct17backup.tar /home** writes a backup of the /home directory to the**/root/tarbackups/oct17backup.tar** file with verbose output. **tar -xvf /root/tarbackups/oct17backup.tar -C /home** extracts the files and decompresses them to the **/home** directory.

	newer than the existing files in an archive. • **-X *file_name*** causes tar to exclude the file names contained in the specified file when creating an archive file.	
gzip	Compress a file using **gzip**. Options include: • **-c** writes the file to standard output. • **-d** decompresses the file. • **-l** displays information about files in an archive. • **-r** recursively compresses all files in directories and subdirectories. This is the same as the **tar -z** command.	**gzip file.tar** compresses an archive file created with tar. The original uncompressed file is removed. **gzip -c file.tar > file.tar.gz** compresses a tar archive, but leaves the original file unchanged. **gzip -d file.tar.gz** decompresses the tar archive.
gunzip	Decompress a file using **gunzip**. Options include: • **-f** forces decompression even if the file has multiple links or the corresponding file already exists. • **-r** decompress all files in a directory tree.	**gunzip file.tar.gz** decompresses the tar archive. **gunzip file.cpio** decompresses the cpio file.
bzip2	Compress and decompress a file using **bzip2**. Options include: • **-z** compresses a file. • **-d** decompresses a file. • **-k** keeps the original file unchanged. This is the same as **tar -j** command.	**bzip2 file.tar** compresses the tar archive file and removes the original file. **bzip2 -k file.tar** compresses the tar archive file, but leaves the original file unchanged. **bzip2 -d file.tar.bz2** decompresses the tar archive.
xz	Compress and decompress a file using **xz**. Options include: • **-z** compresses a file. • **-d** decompresses a file. • **-k** keeps the original file unchanged. This is the same as **tar -J** command.	**xz file.tar** compresses the tar archive file and removes the original file. **xz -k file.tar** compresses the tar archive file, but leaves the original file unchanged. **xz -d file.tar.gz** decompresses the tar archive.

From <https://cdn.testout.com/client-v5-1-10-414/startlabsim.html>

7.12.8

Monday, April 24, 2017 10:51 AM

Be aware of the following concerning the **cpio** and **dd** commands:

Use	To	Example
cpio	Create a **cpio** archive or extract files from a **cpio** archive. The **cpio** command is used as another method to archive files, yet it is different from other archive utilities, because it takes only the files names from standard input. The cpio command: • Copies files to an archive (copy-out mode). • Extracts files from an archive (copy-in mode). • Copies files to a different directory tree (copy-pass mode). The cpio command uses the following options: • **-o** creates the archive in copy-out mode. • **-v** causes cpio to display verbose output, showing file names as they're added or removed. • **-i** extracts files by invoking copy-in mode. • **-u** overwrites existing files. • **-d** creates directory paths (if needed) during extraction. • **-t** displays archive contents without extracting files. • **-p** copies files to a new directory (copy-pass mode).	**ls ~/4archive | cpio -ov > *filename*.cpio** creates a cpio archive from the files in the **~/4archive** directory. **cpio -iv < *filename*.cpio** extracts the files from the cpio archive. **ls ~/copyme | cpio -pvd ./newdirectory** copies files from **~/copyme** to **./newdirectory**.
dd	Copy information using records. The **dd** utility copies information using records instead of files. **dd** is useful for: • Copying partitions to a single image file. • Copying a master boot record. Parameters for **dd** include: • **if=** specifies the input file. • **of=** specifies the output file. • **bs=** specifies the number of bytes to process at a time. • **count=** specifies the number of blocks to be copied.	**dd if=/dev/sdb1 of=/root/partition.image** copies the entire first partition if the second hard drive to a single file. **dd if=/dev/sda of=/root/file.mbr bs=512 count=1** copies the master boot record of the first hard drive to a single file.

From <https://cdn.testout.com/client-v5-1-10-414/startlabsim.html>

8.1.4

Monday, April 24, 2017 10:52 AM

A *device driver* is a software component that allows a hardware device to communicate with the operating system of a computer. Drivers allow an operating system to correctly interpret and implement the signals that come from the hardware device. The following table describes the two methods Linux uses to implement device drivers:

Method	Description
Loaded as a kernel module	A kernel *module* is software that the kernel accesses only when it is needed. When in use, modules run as if they were part of the kernel and have the same access rights. Modules: • Have an .o or .ko extension. • Are stored in the **/lib/modules/***kernel_version***/kernel/drivers/***driver_name* directory. • Are linked and unlinked dynamically.
Compiled into the kernel	When the drivers are compiled into the kernel, they are integrated into the kernel build when the kernel is recompiled. This method requires an administrator to recompile the kernel. Drivers compiled into the kernel: • Increase the size and complexity of the kernel. • Require considerable configuration expertise. • Consume additional computer resources. • Should be limited to the hardware needed to boot the computer, such as drivers for the keyboard, mouse, and disk drive.

The following directories contain information about the hardware that is installed on the computer:

Directory	Contents
/proc	The **/proc** directory contains information about the system state and processes. Its contents are created dynamically. Files and directories found in the **/proc** directory are: • **cmdline** contains the boot options that were given to the kernel at boot time. • **cpuinfo** contains information about the computer's CPU. • **devices** contains a list of hardware installed on the computer. • **dma** contains all the direct memory access assignments for the computer. Direct memory access gives hardware devices direct access to the computer's memory independent of the CPU. • **interrupts** lists the interrupt request (IRQ) channels the computer uses. Interrupt requests are signals sent to the CPU that inform it that it needs to process input from a hardware device. • **iomem** contains a mapping of the memory allocated to each device and the input/output port assignments for the memory. • **modules** lists the kernel modules that the computer is currently using. • **version** gives information about the current kernel version. • **meminfo** displays detailed memory information on the system. • **/scsi** contains a file or directory for each SCSI device attached to the computer. • **/bus** contains a file or directory for each USB device attached to the computer.

	• **/ide** contains a file for the IDE devices attached to the computer, including the internal hard drives and other devices that attach to an IDE ribbon. Be aware of the following facts about the files within **/proc**: • Use the **cat** command (or other text viewing utilities) to view files in the **/proc** directory and subdirectories. • Do not use **vi** to view or modify files in the **/proc** directory. Instead, use the **echo** command to redirect commands to the appropriate files or commands.
/sys	The **/sys** directory displays information about devices and drivers. The following directories are found in the **/sys**directory: • **/block** has an entry for each block device on the computer. Block devices include flash drives and hard disk drives. • **/bus** holds a subdirectory for SCSI, USB, PCI, and ISA devices. Each of these subdirectories has an additional directory for devices and drivers that has information for each device and driver in the category. • **/class** has files for each class of devices on the computer. • **/devices** lists every device that has been discovered on the computer. The directory hierarchy places each device beneath the device to which it is connected. • **/module** has a subdirectory for each kernel module installed on the computer.

Linux includes several utilities that provide extensive information about your hardware configuration, including:

Use	To	Examples
lsusb	Display information on all USB devices connected to the computer. This utility uses the following options: • **-v** shows exhaustive information. • **-s** *bus_name* shows information for a specific bus. • **-t** displays the USB device hierarchy as a tree.	**lsusb -v** shows all information about each USB device on the computer.
hwinfo	Display information about hardware on the computer. Be aware of the following options: • **--*hardware_item_name*** probes for a specific hardware item. Common hardware names include: • **bluetooth** • **camera** • **cdrom** • **cpu** • **disk** • **dsl** • **monitor** • **mouse** • **keyboard** • **usb** • **--short** shows an abbreviated list of information. • **--listmd** displays RAID devices. Not all distributions include the hwinfo command.	**hwinfo --cpu** shows information about the computer's CPU.
lspci	Display information for all PCI devices. Be aware of the following options:	**lspci -k** shows the devices and the kernel drivers that support

	• **-k** shows the kernel drivers that support the device. • **-t** displays a tree diagram that shows connections between all busses, bridges, and devices.	them.

From <https://cdn.testout.com/client-v5-1-10-414/startlabsim.html>

8.2.3

Monday, April 24, 2017 10:53 AM

When the system boots, it uses one of the following files to automatically load kernel modules. (The exact file used depends on the implementation.)

File	Description
/etc/modprobe.conf	Provides the **modprobe** utility with default commands for loading modules at boot time. Entries in the file include the following: • **install** loads a module at boot time. • **alias** specifies a name as an alias for a module name. This alias can be used with module utilities. • **options** specifies options used while loading a module, including: • **irq** for IRQ information • **io** for I/O port information.
/etc/modprobe.d	Contains multiple configuration files used by **modprobe** at boot time if the **/etc/modprobe.conf** file does not exist.

You can use the following commands to manage kernel modules manually:

Use	To	Example
lsmod	List all loaded modules. The command pulls information from the **/proc/modules** file.	
cat /proc/modules	View the **/proc/modules** file. This file contains a list of all loaded modules.	
modinfo	View additional information about a module listed using the **lsmod** command.	**modinfo mii** shows information about the MII Hardware Support Library module.
depmod	Create a file that identifies module dependencies. The file is placed at**/lib/modules/***kernel_version_number***/modules.dep**. This command first reads the **/etc/modules.conf** file to identify modules. It then probes each module to build a list of dependencies. Be aware of the following options: • **-a** shows information for all modules. • **-n** shows what would happen on the screen, but does not perform the action. • **-v** uses verbose mode.	**depmod -an** performs the probe and display the results on the screen. **depmod -v** displays all module information to the screen as it updates the modules.dep file.
insmod	Install a module. • This command does not look for dependencies. It will fail if a module being loaded has unresolved dependencies. • Include the full name of the module, including the .o or .ko extension.	**insmod mousedev.ko** loads the mousedev module.
modprobe	Load a module along with any module dependencies. This utility also runs at startup to load modules.	**modprobe reiserfs** loads the

	The **/etc/modprobe.conf** file provides **modprobe** with its configuration information. Be aware of the following options: • **-l** lists all loaded modules. • **-r** removes a module. This option checks for dependencies before unloading the module.	reiserfs and all of its dependent modules. **modprobe -r reiserfs** removes the reiserfs module.
rmmod	Remove a module from the kernel. **rmmod**: • Cannot unload the module if it is in use. • Does not look for dependencies and can cause errors if a module depends on a module that is unloaded.	**rmmod mousedev** removes the mousedev module.

From <https://cdn.testout.com/client-v5-1-10-414/startlabsim.html>

8.3.2

Monday, April 24, 2017 10:53 AM

Be aware of the following device categories when managing hardware:

Category	Description
Coldplug	Coldplug devices should only be removed or replaced when the power to the computer is off. Attempting to remove these devices while the power is on can damage the device or the computer. Coldplug devices include: • RAM (Random Access Memory) chips • CPU (Central Processing Unit) • Expansion cards, such as Peripheral Component Interconnect (PCI) or PCI Express cards • Standard hard disk drives
Hotplug	Hotplug devices can be removed while the computer is on. Linux uses software designed to detect these changes as the devices are added and removed. Hotplug devices include: • USB devices • FireWire devices • Hot-swappable hard disk drives

Linux uses the following components to manage devices:

Component	Description
sysfs	*sysfs* creates a virtual file system mounted at **/sys** which exports information about hotplug devices so that other utilities can access the information.
Hardware Abstraction Layer (HAL) daemon	The HAL daemon (hald) provides all running processes with data about current hardware. hald runs constantly.
Desktop Bus (D-Bus) daemon	The D-Bus daemon notifies running processes whenever a hotplug device is connected or disconnected from the system.
udev	The udev daemon (udevd) creates a virtual file system that is mounted at /dev. It communicates with the Linux kernel through the uevent interface. When a hotplug device is added or removed, the kernel sends out a uevent message that is picked up by udevd. Based on the rules defined in the files in the **/etc/udev/rules.d** directory, udevd then: • Initializes the device. • Creates the appropriate device file in the /dev directory. • Configures the device using the **ifup** utility if the new device is a network interface. • Mounts the device using the information in **/etc/fstab** if the new device is a storage device. • Informs running processes about the new device.

From <https://cdn.testout.com/client-v5-1-10-414/startlabsim.html>