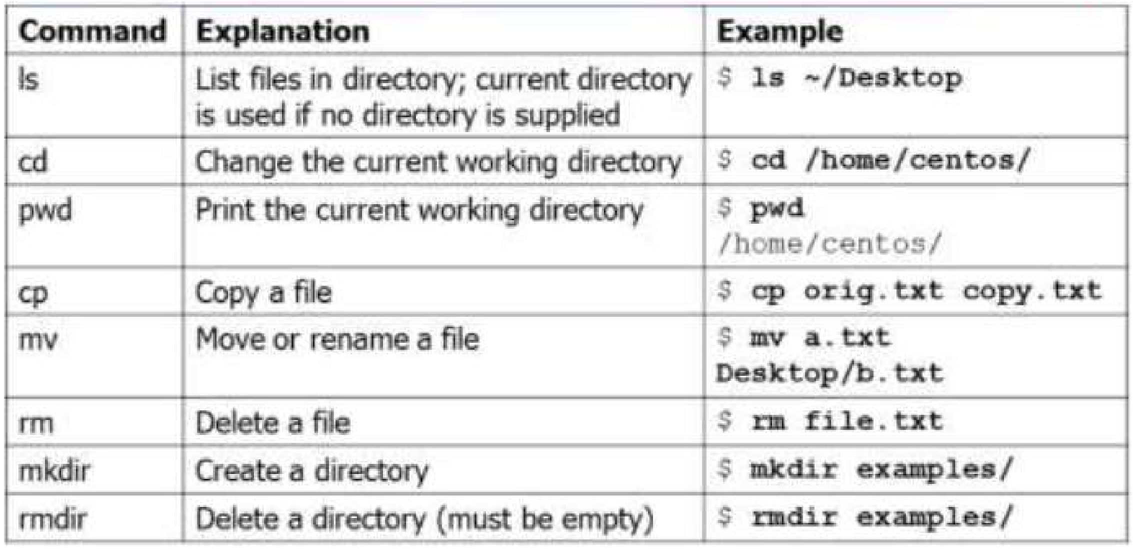
Linux Commands

Friday, October 2, 2020 12:17 PM



Command	Explanation	Example
ls	List files in directory; current directory is used if no directory is supplied	`$ ls ~/Desktop`
cd	Change the current working directory	`$ cd /home/centos/`
pwd	Print the current working directory	`$ pwd` `/home/centos/`
cp	Copy a file	`$ cp orig.txt copy.txt`
mv	Move or rename a file	`$ mv a.txt Desktop/b.txt`
rm	Delete a file	`$ rm file.txt`
mkdir	Create a directory	`$ mkdir examples/`
rmdir	Delete a directory (must be empty)	`$ rmdir examples/`

Important Commands (1)

ls (list directory contents): Lists files in a directory. The directory to list can be specified as a parameter, otherwise it lists the current directory.

cd (change directory): Change the current working directory to the one specified. You can specify either an absolute path (such as `cd /home/centos/`) or a relative path (such as `cd centos` or `cd ..`).

pwd (print working directory): Print the current working directory (the directory you are currently in).

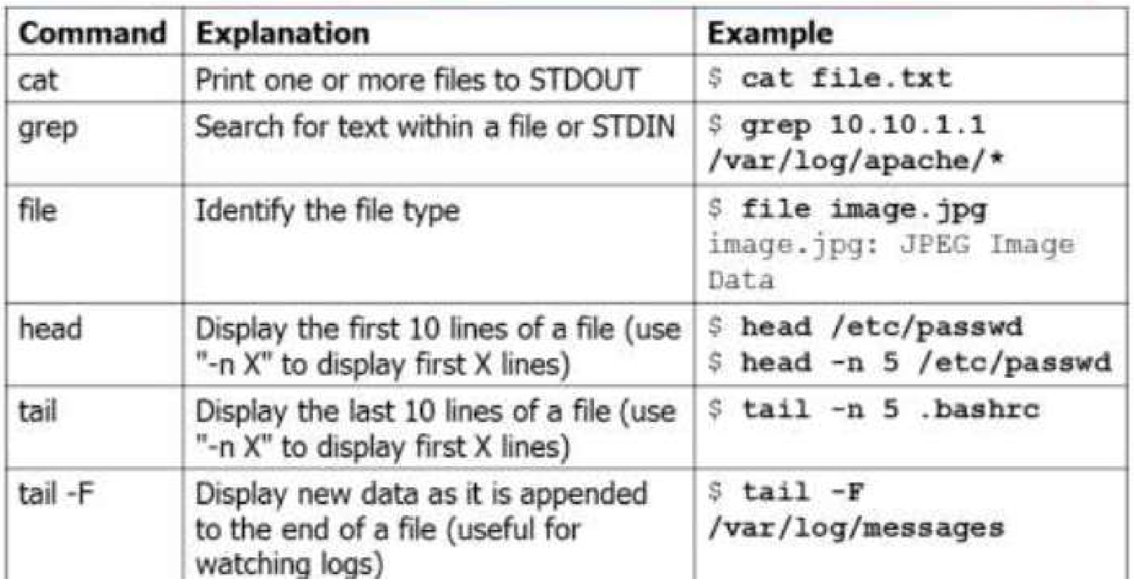
cp (copy): Copy files or directories. To copy many files at once, specify them each as a parameter or use a wildcard (such as `*.txt`), and specify the destination directory as the final parameter. To copy a directory and its contents, use the "-R" (recursive) option, as in: `cp -R dir1/ dir2/`

mv (move): Move or rename files or directories. Like "cp", you can move multiple files at once by specifying each one or using a wildcard and then specifying the destination directory as the final parameter. "mv" is also used to rename files or directories by moving them from their old name to their new name.

rm (remove): Remove (delete) a file.

mkdir (make directories): Create a directory. The "-p" option can be used to create any parent directories needed to create the final directory in one command. For example, `mkdir -p /dir1/dir2/dir3` will create "/dir1" if it does not already exist, then create "/dir1/dir2" if it does not already exist, and finally create "/dir1/dir2/dir3" if it does not already exist. If any of them did exist, no errors will be returned.

rmdir (remove empty directories): Removes (deletes) an empty directory. It also supports "-p" to remove a directory and all its subdirectories. For example, `rmdir -p /dir1/dir2/dir3` would first delete "/dir1/dir2/dir3", then delete "/dir1/dir2", and finally delete "/dir1".



Command	Explanation	Example
cat	Print one or more files to STDOUT	`$ cat file.txt`
grep	Search for text within a file or STDIN	`$ grep 10.10.1.1 /var/log/apache/*`
file	Identify the file type	`$ file image.jpg` `image.jpg: JPEG Image Data`
head	Display the first 10 lines of a file (use "-n X" to display first X lines)	`$ head /etc/passwd` `$ head -n 5 /etc/passwd`
tail	Display the last 10 lines of a file (use "-n X" to display first X lines)	`$ tail -n 5 .bashrc`
tail -F	Display new data as it is appended to the end of a file (useful for watching logs)	`$ tail -F /var/log/messages`

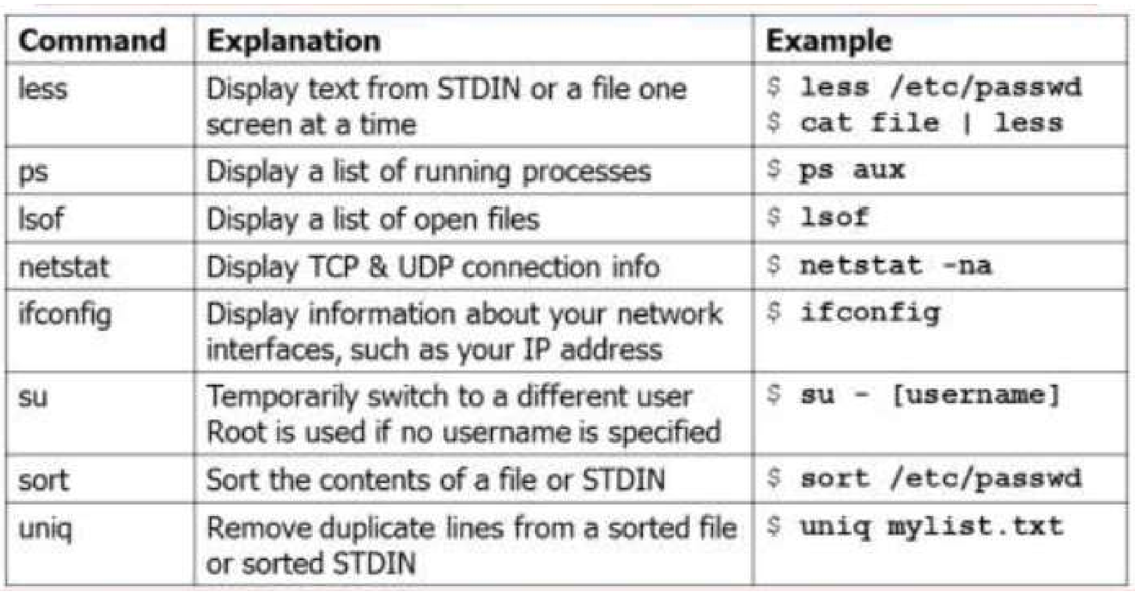
cat (concatenate): Print one or more files and print to STDOUT. It's called "concatenate" because it can be used to combine multiple files into one stream, such as: `cat *.log > combined-log`.

grep (global regular expression print): Searches for text within a file or STDIN. It is commonly used as part of a pipeline instead of directly on a file. For example, to search for established network connections, you could run `netstat -nat | grep ESTABLISHED`. It can be used to search for a regular expression pattern instead of simple text, which is beyond the scope of this introduction.

file: Identifies the file type by inspecting the file's contents, particularly its header information. It has a database of "magic numbers" that correspond to various file types.

head: Displays the first X lines of a file, where X is ten by default. The number of lines can be specified using the "-n" option, and "-c" can be used to specify a number of bytes to display instead of lines.

tail: Displays the last X lines of a file, where X is ten by default. The number of lines can be specified using the "-n" option, and "-c" can be used to specify a number of bytes to display instead of lines. tail also has a "follow" option ("-F") which outputs new data as it is appended to the end of a file. This is convenient for watching log files in real-time.



Command	Explanation	Example
less	Display text from STDIN or a file one screen at a time	`$ less /etc/passwd` `$ cat file | less`
ps	Display a list of running processes	`$ ps aux`
lsof	Display a list of open files	`$ lsof`
netstat	Display TCP & UDP connection info	`$ netstat -na`
ifconfig	Display information about your network interfaces, such as your IP address	`$ ifconfig`
su	Temporarily switch to a different user Root is used if no username is specified	`$ su - [username]`
sort	Sort the contents of a file or STDIN	`$ sort /etc/passwd`
uniq	Remove duplicate lines from a sorted file or sorted STDIN	`$ uniq mylist.txt`

less: Display text from STDIN or a file one screen at a time, making it easier to read. less is commonly used to pipe the output of commands into, particularly commands with a lot of output. The name "less" is a joke on the pager "more", which less is an improved version of (less is more).

ps: Display information about running processes. The "aux" options indicate to list all running processes in the system, instead of only processes from the current shell. Output from "ps" is often piped into grep to search for instances of a particular program.

lsof (list open files): Displays a list of open files on the system. The list includes details on which user and which process has the file open. lsof also has a "-i" option that will display a list of programs that are listening for network traffic, similar to the output of `netstat -na | grep LISTEN`.

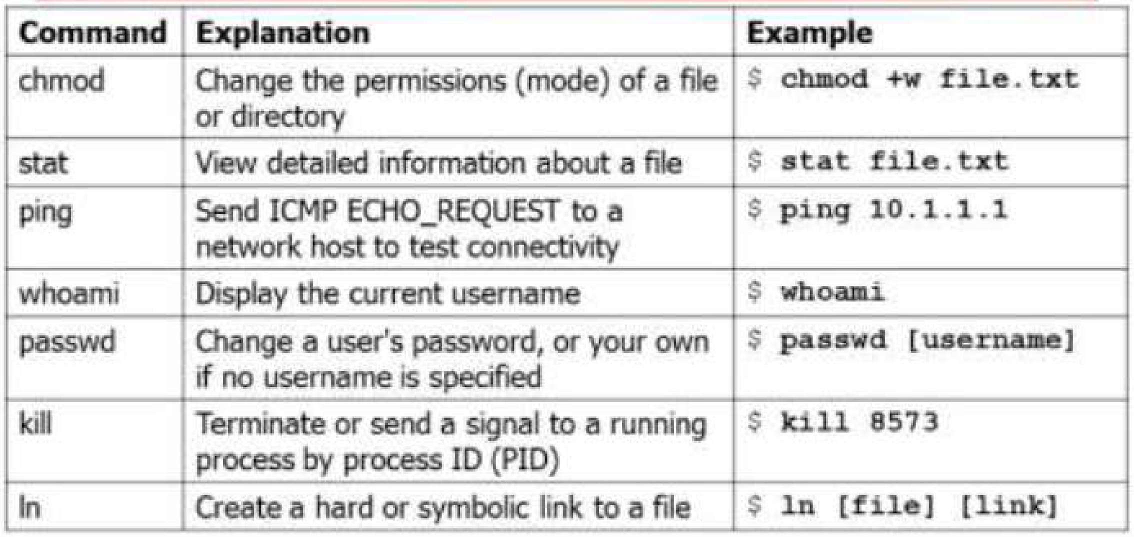
netstat: Displays information about TCP and UDP connections on the system, including established connections and ports that have services listening for incoming connections. The "-t" and "-u" options can be used to show only TCP or UDP information, respectively.

ifconfig (interface config): Displays information about your network interfaces, such as your IP address (similar to "ipconfig" on Windows). ifconfig can also be used to set the IP address for an interface.

su (substutute user): Temporarily switch to a different user (most commonly root). The "-" option makes the shell a login shell, causing it to inherit the target user's environment.

sort: Sort the contents of a file or STDIN. The data is sorted alphabetically by default, but can also be sorted numerically with the "-n" option.

uniq: Remove duplicate lines from a sorted file or sorted STDIN (you must pipe through "sort" first).



Command	Explanation	Example
chmod	Change the permissions (mode) of a file or directory	`$ chmod +w file.txt`
stat	View detailed information about a file	`$ stat file.txt`
ping	Send ICMP ECHO_REQUEST to a network host to test connectivity	`$ ping 10.1.1.1`
whoami	Display the current username	`$ whoami`
passwd	Change a user's password, or your own if no username is specified	`$ passwd [username]`
kill	Terminate or send a signal to a running process by process ID (PID)	`$ kill 8573`
ln	Create a hard or symbolic link to a file	`$ ln [file] [link]`

chmod (change mode): Change the permissions (mode) of a file or directory. More information on chmod and permissions can be found in an upcoming section.

stat: View detailed information about a file, including its name, size, last modified date, and permissions.

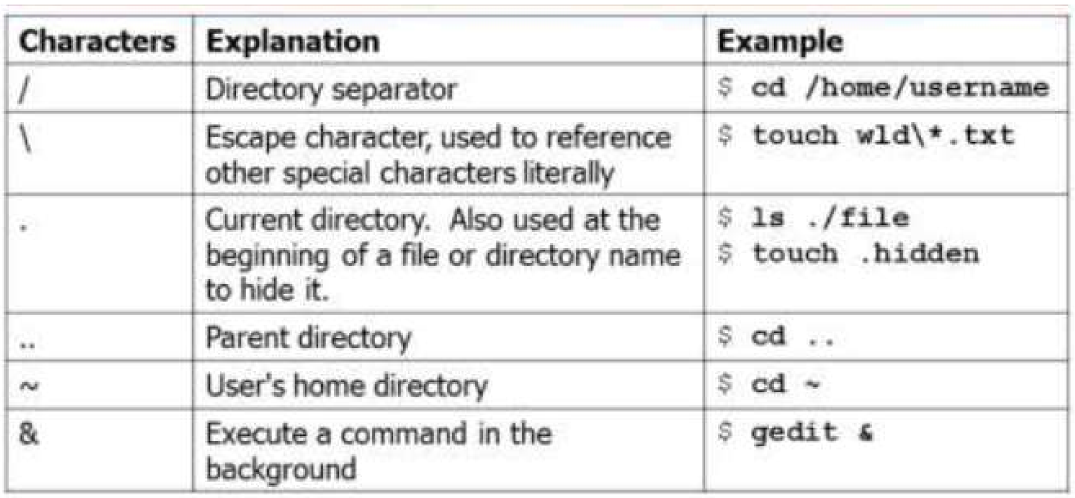
ping: Send ICMP ECHO_REQUEST packets to a network host and wait for responses to test network connectivity. The "ping6" program can be used for IPv6 addresses.

whoami: Display the current username (the username the "whoami" program is running as, normally the user who invoked it).

passwd: Change a password. With no options, it is used to change your own password. The root user can specify a username as a parameter to change that user's password.

kill: Terminate or send a signal (such as "HUP") to a running process. This is most commonly used to kill a running process by PID, but can also be used to send arbitrary signals to a process. The "HUP" signal is commonly used to restart a process.

ln (link): Create a hard or symbolic link to a file. A hard link is a separate file listing that points to the same data on the disk. A symbolic link is a special file that contains the path to the file the link is pointing to. A symbolic link can point to a directory, but a hard link cannot. To create a symbolic link, use the "-s" option.



Characters	Explanation	Example
/	Directory separator	`$ cd /home/username`
\	Escape character, used to reference other special characters literally	`$ touch wld\*.txt`

.	Current directory. Also used at the beginning of a file or directory name to hide it.	`$ ls ./file`
		`$ touch .hidden`

..	Parent directory	`$ cd ..`
~	User's home directory	`$ cd ~`
&	Execute a command in the background	`$ gedit &`

These special characters have special meaning in the shell:

/ **(forward slash):** Directory separator (used between directory names in a file path).

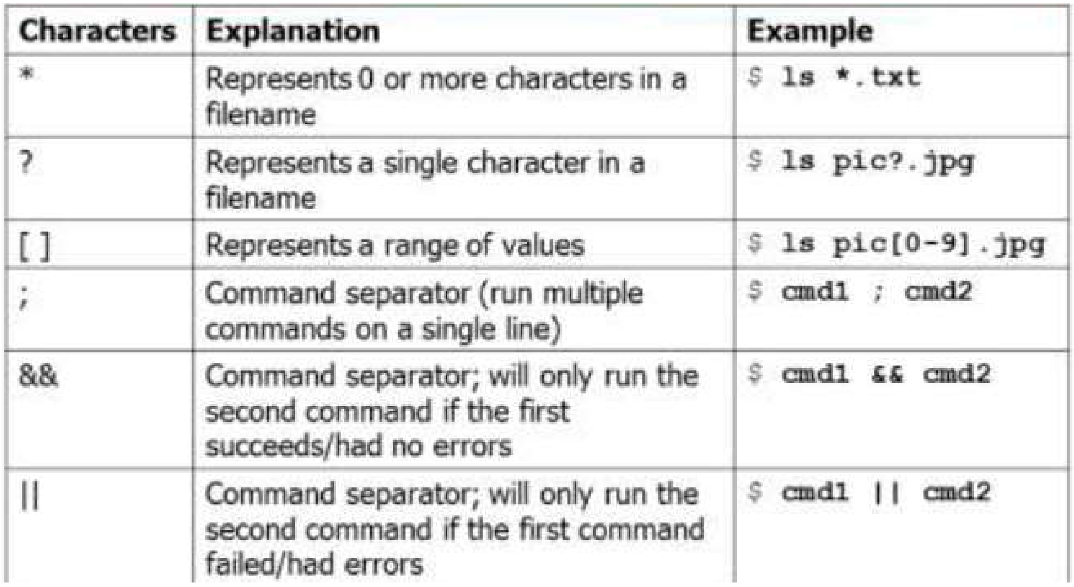
\ **(backslash):** This is the escape character, which is used to reference other special characters literally. In other words, if you need to use a special character as itself in a command, put a backslash in front of it to cause the shell to interpret it literally.

. **(single dot):** This represents the current directory. It is also used as the first character of a file or directory name to mark it as hidden.

.. **(two dots):** This represents the parent directory, one level up from the current directory.

~ **(tilde):** This represents the current user's home directory, and can be used as shorthand for it. For example, `cd ~/Desktop` could be used as shorthand for `cd /home/username/Desktop`. The tilde can also have a username immediately after it to substitute that user's home directory instead of your own, such as `cd ~otheruser/Desktop`.

& **(ampersand):** This is used to execute a command in the background as a job. When you run a command in the background, you will get a shell prompt back right away instead of having to wait for the command to finish.



Characters	Explanation	Example
*	Represents 0 or more characters in a filename	`$ ls *.txt`
?	Represents a single character in a filename	`$ ls pic?.jpg`
[ ]	Represents a range of values	`$ ls pic[0-9].jpg`
;	Command separator (run multiple commands on a single line)	`$ cmd1 ; cmd2`
&&	Command separator; will only run the second command if the first succeeds/had no errors	`$ cmd1 && cmd2`
||	Command separator; will only run the second command if the first command failed/had errors	`$ cmd1 || cmd2`

*** (asterisk):** A wildcard used to represent zero or more characters in a filename. For example, `ls *.txt` will list the names of any files ending in ".txt", such as "file1.txt" and "file23.txt".

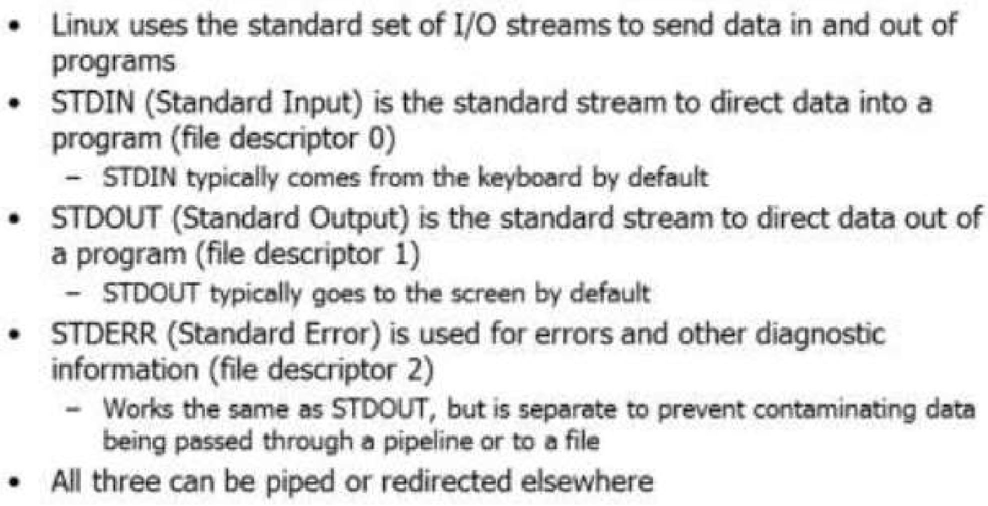
? (question mark): A wildcard used to represent a single character in a filename. For example, running `ls pic?.jpg` would match "pic1.jpg" and "pic2.jpg", but not "pic23.jpg" or "pic.jpg".

[] (square brackets): These are used to specify a range of values to match. For example, "[0-9]" would match any digit 0 through 9, and "[a-z]" would match any lowercase letter.

; (semicolon): A command separator that can be used to run multiple commands on a single line, unconditionally.

&& (double ampserand): A command separator, which will only run the second command if the first command is successful (does not return an error). This is commonly used in shell scripts.

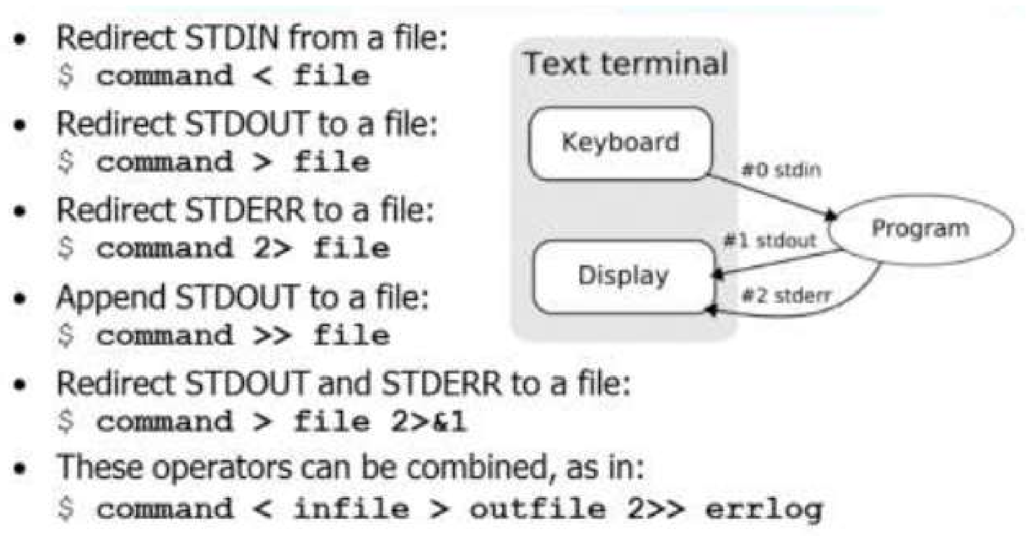
|| (double pipe): A command separator, which will only run the second command if the first command failed (had errors). This is commonly used in shell scripts, such as to terminate the script if an important command fails.



- Linux uses the standard set of I/O streams to send data in and out of programs
- STDIN (Standard Input) is the standard stream to direct data into a program (file descriptor 0)
  - STDIN typically comes from the keyboard by default
- STDOUT (Standard Output) is the standard stream to direct data out of a program (file descriptor 1)
  - STDOUT typically goes to the screen by default
- STDERR (Standard Error) is used for errors and other diagnostic information (file descriptor 2)
  - Works the same as STDOUT, but is separate to prevent contaminating data being passed through a pipeline or to a file
- All three can be piped or redirected elsewhere

STDIN, STDOUT, and STDERR

Linux uses the standard set of I/O (input/output) streams to send data in and out of programs. STDIN (Standard Input) is the standard stream to direct data into a program, typically from the keyboard by default. STDIN has file descriptor 0. STDOUT (Standard Output) is the standard stream to direct data out of a program, typically to the screen by default. STDOUT has file descriptor 1. STDERR (Standard Error) is used for errors and other diagnostic information, and has file descriptor 2. STDERR works the same as STDOUT, but is a separate stream to prevent contamination data being passed through the pipeline to another program or to a file. All three streams can be piped or redirected elsewhere.



- Redirect STDIN from a file:
  ```
  $ command < file
  ```
- Redirect STDOUT to a file:
  ```
  $ command > file
  ```
- Redirect STDERR to a file:
  ```
  $ command 2> file
  ```
- Append STDOUT to a file:
  ```
  $ command >> file
  ```
- Redirect STDOUT and STDERR to a file:
  ```
  $ command > file 2>&1
  ```
- These operators can be combined, as in:
  ```
  $ command < infile > outfile 2>> errlog
  ```

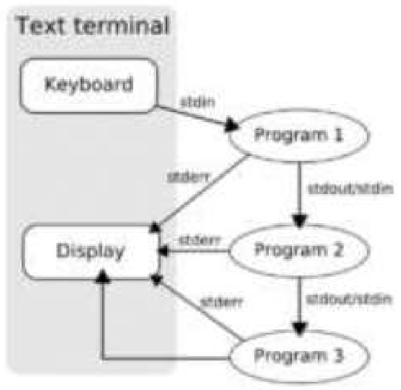
Redirection

Redirection allows you to redirect the standard I/O streams to different locations, such as to a file or a pipe. For example, you can redirect STDIN to read data from a file instead of from the keyboard, redirect STDOUT to write to a file instead of the screen, and redirect STDERR to hide its output (such as by sending it to /dev/null, a black hole that discards any data it receives). Here are some examples:

Redirect STDIN from a file:
```
$ command < file
```
Redirect STDOUT to a file:
```
$ command > file
```
Redirect STDERR to a file (note the file descriptor "2"):
```
$ command 2> file
```
Append STDOUT to a file (write STDOUT to the end of an existing file):
```
$ command >> file
```
Redirect STDOUT and STDERR to a file (the "2>&1" sends 2 to file descriptor 1, which is STDOUT):
```
$ command > file 2>&1
```
These operators can be combined, as in:
```
$ command < infile > outfile 2>> errlog
```
The above command would receive input from "infile", save the output to "outfile" (overwriting "outfile" if it already exists), and append any error messages to "errlog".

- Pipes are used to connect the STDOUT of one program to the STDIN of another
  - STDERR is still sent to the display unless otherwise redirected
- The pipe character, "|", is used between commands
- Pipes are often used in conjunction with with other commands, such as grep, to filter the output of commands
- Example:

```
$ cat /etc/passwd | grep :0: | sort
```



Pipes

Pipes are used to connect the STDOUT of one program to the STDIN of another, creating a pipeline for data to flow through a series of programs. To use pipes, place the pipe character ("|", or shift-\) between commands. Pipes are often used to filter the output of commands, such as to search for a particular string, or to sort a set of data. The programs commonly used for these tasks (such as "grep", "sort", and "uniq") are often referred to as filters.

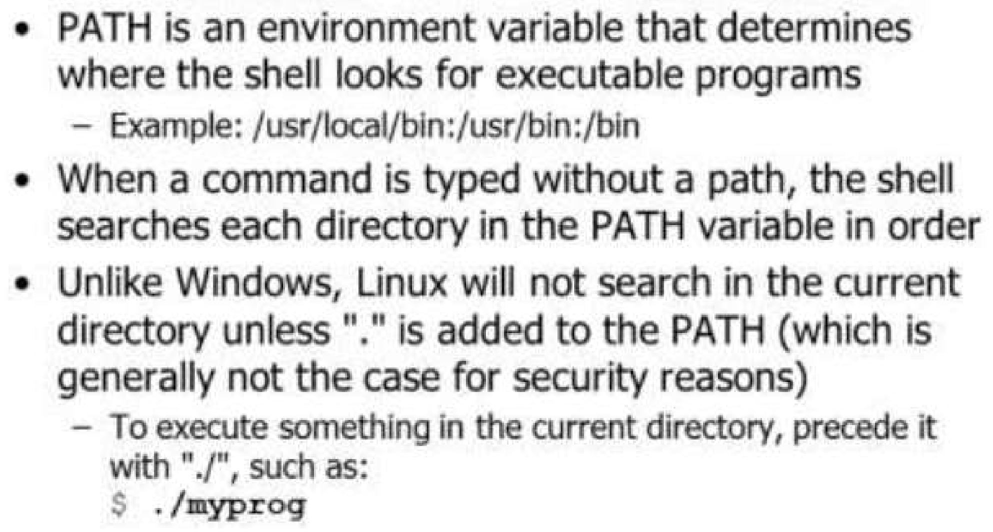
For example, the following command will read in the list of users on the system, search them for the string ":0:" (identifying users with UID or GID 0), and then sort them alphabetically:

```
$ cat /etc/passwd | grep :0: | sort
```

Note that STDERR is still sent to the display unless it is redirected elsewhere. This allows you to see any errors or warnings that may occur without them becoming part of the pipeline. If you want to redirect STDERR into STDOUT you can use this syntax:

```
$ command 2>&1
```

In the above command STDERR (2) is redirected into (>) file descriptor 1, STDOUT (&1).



- PATH is an environment variable that determines where the shell looks for executable programs
  - Example: /usr/local/bin:/usr/bin:/bin
- When a command is typed without a path, the shell searches each directory in the PATH variable in order
- Unlike Windows, Linux will not search in the current directory unless "." is added to the PATH (which is generally not the case for security reasons)
  - To execute something in the current directory, precede it with "./", such as:

```
$ ./myprog
```

PATH

"PATH" is an environment variable that determines where the shell looks for executable programs. When a command is typed without a path to the executable, the shell searches each directory in the PATH variable in order. For example, if the PATH variable is set to "/usr/local/bin:/usr/bin:/bin" and you type the command "ls", the shell will first check "/usr/local/bin" for the executable program "ls", then check /usr/bin, and then

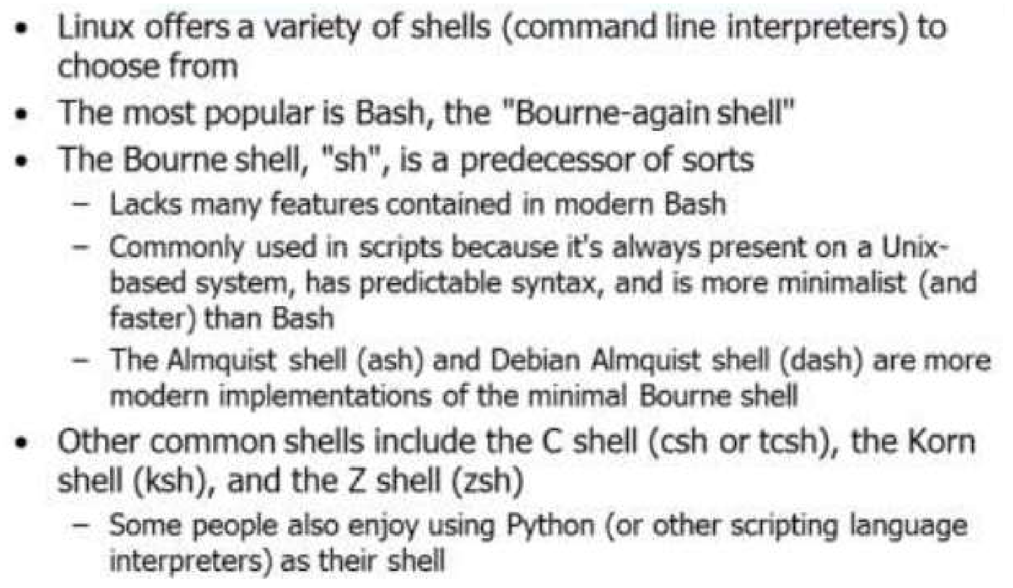
PATH

"PATH" is an environment variable that determines where the shell looks for executable programs. When a command is typed without a path to the executable, the shell searches each directory in the PATH variable in order. For example, if the PATH variable is set to "/usr/local/bin:/usr/bin:/bin" and you type the command "ls", the shell will first check "/usr/local/bin" for the executable program "ls", then check /usr/bin, and then check /bin, running whichever one it finds first.

Unlike Windows, Linux will not search the current directory for a command you type unless "." (the current directory) is added to the PATH variable. This is not considered good security practice since it could cause someone to be tricked into running a malicious version of a program, so Linux systems do not include "." in the PATH variable by default. This prevents a malicious user from putting a malicious executable named "pwd" in a directory and causing another user to accidentally run it when the user types **pwd**, as the pwd executable will be run from the path, not the current directory.

If you do need to run an executable that is in the current directory, but not in the path, precede it with "./".

```
$ ls
myprog
$ myprog
bash: myprog: command not found
$ ./myprog
Thank you for running myprog
```

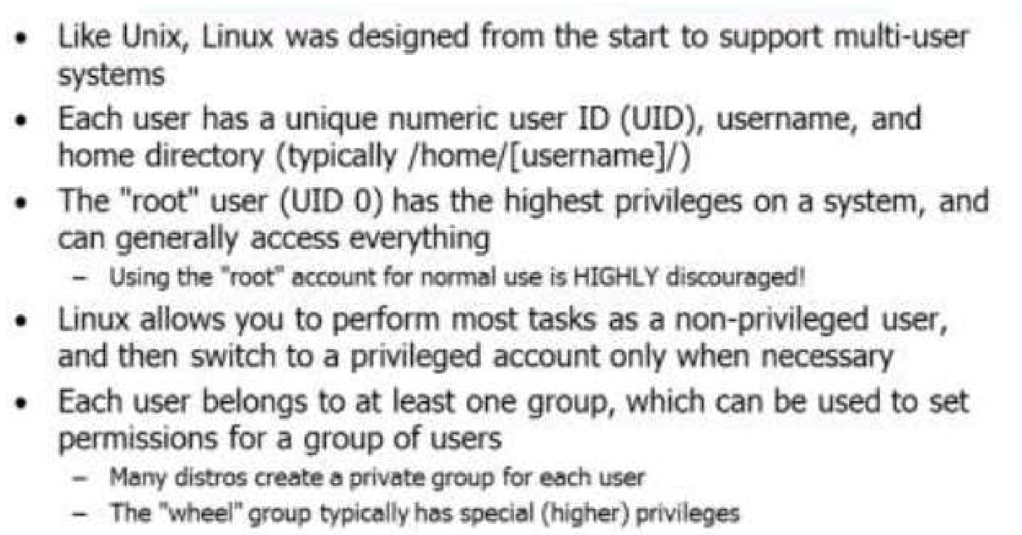


- Linux offers a variety of shells (command line interpreters) to choose from
- The most popular is Bash, the "Bourne-again shell"
- The Bourne shell, "sh", is a predecessor of sorts
  - Lacks many features contained in modern Bash
  - Commonly used in scripts because it's always present on a Unix-based system, has predictable syntax, and is more minimalist (and faster) than Bash
  - The Almquist shell (ash) and Debian Almquist shell (dash) are more modern implementations of the minimal Bourne shell
- Other common shells include the C shell (csh or tcsh), the Korn shell (ksh), and the Z shell (zsh)
  - Some people also enjoy using Python (or other scripting language interpreters) as their shell

Command Shells

Linux offers a very wide array of command line interpreters for users to choose from. Referred to as command shells, each individual interpreter possesses different fortes and offers different features in an attempt to provide the best user experience. Scripting opportunities also vary from shell to shell. While many UNIX shells exist, the default (and most popular) shell for most Linux distributions is known as Bash, which is an acronym for "Bourne-again shell." The Bourne shell, known simply as "sh", is a predecessor of sorts. The Bourne shell lacks many features contained in modern shells like Bash, but is commonly used in scripts because it's always present on a Unix-based system, and its features and syntax never change. The Almquist shell (ash) and the Debian Almquist shell (dash) are more modern implementations of the minimal Bourne shell.

A comprehensive comparison of UNIX shells (also including Windows and various programming shells) is available at https://en.wikipedia.org/wiki/Comparison_of_command_shells.



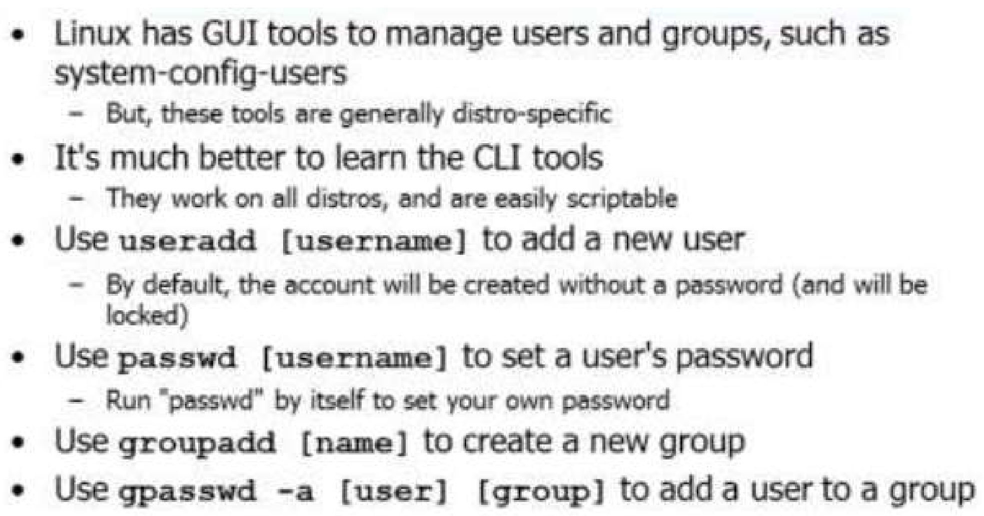
- Like Unix, Linux was designed from the start to support multi-user systems
- Each user has a unique numeric user ID (UID), username, and home directory (typically /home/[username]/)
- The "root" user (UID 0) has the highest privileges on a system, and can generally access everything
  - Using the "root" account for normal use is HIGHLY discouraged!
- Linux allows you to perform most tasks as a non-privileged user, and then switch to a privileged account only when necessary
- Each user belongs to at least one group, which can be used to set permissions for a group of users
  - Many distros create a private group for each user
  - The "wheel" group typically has special (higher) privileges

Users and Groups

Like Unix, Linux was designed from the start to support multi-user systems. Each user can independently use the system without affecting other users. Each user has a unique numeric user ID (UID), a username, and a home directory (typically /home/[username]) where their files and settings are stored. It's worth noting that it is possible for users to share a UID (most commonly to have multiple root-level users), but it is not good practice.

The "root" user (UID 0) has the highest privileges on a system, and can generally access and manipulate everything. Therefore, using the "root" account for anything other than administrative tasks that require it is strongly discouraged. For example, using root to browse the web is a bad idea because if someone exploits a vulnerability in your web browser, they now have full root privileges on your system. It is also very easy to make a mistake when using the command line, so staying out of the root account helps you avoid letting your typos have a system-wide impact. Linux and Linux-based software is generally designed such that you can perform most tasks without needing root access, and it's easy to obtain root privileges when you need them.

Linux also supports groups, which allow permissions or privileges to be assigned to sets of users. Many Linux distributions create a "private" group for each user, which is simply a group with the same name as the user's username, where the user is the only member. Users in the "wheel" group typically have special (i.e., higher) privileges on the system, such as the ability to become root, or the ability to bypass certain security restrictions.



- Linux has GUI tools to manage users and groups, such as system-config-users
  - But, these tools are generally distro-specific
- It's much better to learn the CLI tools
  - They work on all distros, and are easily scriptable
- Use **`useradd [username]`** to add a new user
  - By default, the account will be created without a password (and will be locked)
- Use **`passwd [username]`** to set a user's password
  - Run "passwd" by itself to set your own password
- Use **`groupadd [name]`** to create a new group
- Use **`gpasswd -a [user] [group]`** to add a user to a group

User Management

Linux provides GUI tools to manage user accounts, but these tools are generally unique to each individual distribution. For example, Red Hat comes with system-config-users, Suse comes with Yast, and Debian Linux and other distros come with the users-admin GUI tool set. Instead of learning how to use different tools for different distributions, you should instead learn how to use the command line to perform these same actions on any distribution. By utilizing the command line, you are also able to easily automate user management tasks through shell scripting.

To add a new user to a system, use the "useradd" command, passing the username as the parameter. For example, to create a new user account called "testuser", run:

```
# useradd testuser
```

By default, new user accounts are created without a password (null, not a blank password) and are locked to prevent someone logging in. To set testuser's password, use the "passwd" command:

```
# passwd testuser
Changing password for user testuser.
New password: [enter password here]
Retype new password: [enter password here]
passwd: all authentication tokens updated successfully.
```

To create a new group, use the "groupadd" command:

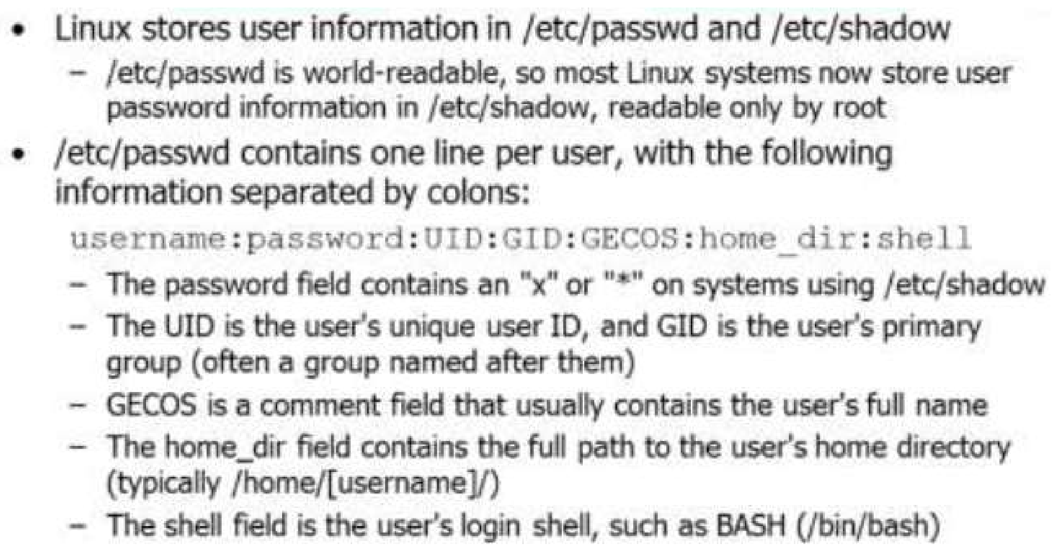
```
# groupadd testgroup
```

To add testuser to the new "testgroup" group, use the "gpasswd" command:

```
# gpasswd -a testuser testgroup
```

For a comprehensive introduction to users, please see the Chapter 32 in the CentOS Deployment Guide:

http://www.centos.org/docs/6/html/Deployment_Guide-en-US/ch-users-groups.html



- Linux stores user information in /etc/passwd and /etc/shadow
  - /etc/passwd is world-readable, so most Linux systems now store user password information in /etc/shadow, readable only by root
- /etc/passwd contains one line per user, with the following information separated by colons:

```
username:password:UID:GID:GECOS:home_dir:shell
```

  - The password field contains an "x" or "*" on systems using /etc/shadow
  - The UID is the user's unique user ID, and GID is the user's primary group (often a group named after them)
  - GECOS is a comment field that usually contains the user's full name
  - The home_dir field contains the full path to the user's home directory (typically /home/[username]/)
  - The shell field is the user's login shell, such as BASH (/bin/bash)

/etc/passwd

Linux passwords are managed through a joint effort of the /etc/passwd and /etc/shadow files, which provide the operating system a means of storing account information and credentials. Shadow passwords are critical to system security and have thus been implemented in every modern mainstream Linux distribution. It is important for system administrators and attackers alike to understand their purpose and significance. The /etc/passwd file has to be readable by all users on the system because various utilities need to look up user information, so shadow passwords were implemented to keep the passwords private in the protected /etc/shadow file (that only root can read).

The file /etc/passwd contains one line per user, with 7 fields of information separated by colons (:). The fields are:

username: The account's unique username. It can only contain lower case letters, numbers, underscores, and hyphens. It can start with either a lowercase letter or an underscore, and can optionally end with a dollar sign ($).

password: Contains either an "x" if the system is using shadow passwords, or the hash of the user's password if it isn't. If it starts with an exclamation point, the account is disabled.

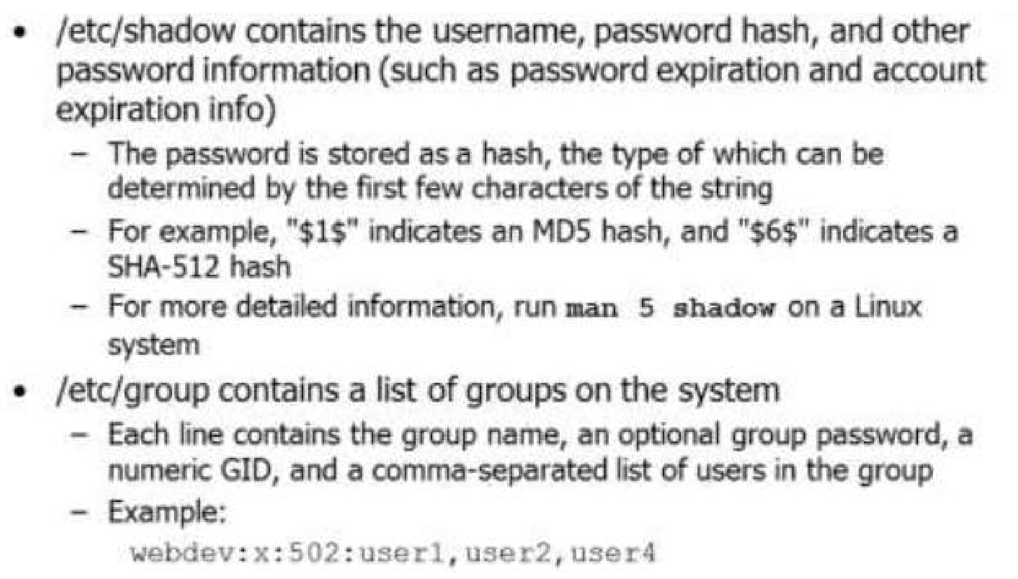
UID: The user's unique user ID, such as "501". The root account is UID 0, and any other account with UID 0 automatically has root privileges.

GID: The GID of the user's primary group. This is the primary group the user will be a member of for the purposes of file permissions, particularly on newly created files.

GECOS: This is used as a comment field, and usually contains the user's full name. It can also contain other information, such as phone numbers. This field can be edited using the "chfn" command.

home_dir: This field contains the full path to the user's home directory, typically /home/[username]/.

shell: This is the user's login shell, such as BASH. If this is not a valid shell, the user will not be able to log in for interactive sessions.



- /etc/shadow contains the username, password hash, and other password information (such as password expiration and account expiration info)
  - The password is stored as a hash, the type of which can be determined by the first few characters of the string
  - For example, "1" indicates an MD5 hash, and "6" indicates a SHA-512 hash
  - For more detailed information, run `man 5 shadow` on a Linux system
- /etc/group contains a list of groups on the system
  - Each line contains the group name, an optional group password, a numeric GID, and a comma-separated list of users in the group
  - Example:

```
webdev:x:502:user1,user2,user4
```

/etc/shadow and /etc/group

On systems with shadow passwords enabled (the vast majority of Linux systems today), the /etc/shadow file contains additional information about each user account, most importantly the password hash. Here is a full breakdown of the colon-separated fields:

Username: The username of the account.

Password: The password hash, which can be in a number of formats supported by crypt(). The hash type is determined by the number found after the first dollar sign. For example, "1" indicates salted MD5, while 6" indicates salted SHA-512.

Date of last password change: The date the user last changed his or her password, expressed as the number of days since January 1, 1970.

Minimum password age: The minimum number of days the user has to wait before changing his or her password after a password change.

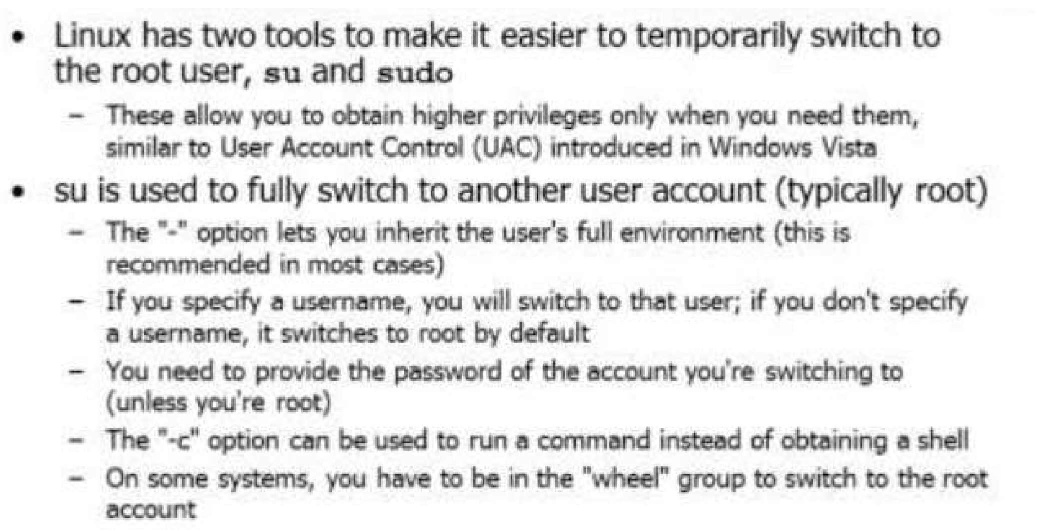
Maximum password age: The maximum number of days the user can keep a password before being forced to change it.

Password warning period: The number of days before a password expires that the user should be warned.

Password inactivity period: The number of days after a password has expired that a password should still be accepted as valid.

Account expiration date: The date that the account will expire and not allow ANY type of login, expressed as the number of days since January 1, 1970.

If different users need the same kind of privileges to certain resources, they can be put into a group and the group given permission to the resource. This prevents from having to assign privileges to each and every account. Group membership is stored in the file /etc/group. Just as there is a root user account with UID 0, there is also a root group with GID 0. Attackers will do anything they can to get UID 0 or GID 0.



- Linux has two tools to make it easier to temporarily switch to the root user, **su** and **sudo**
  - These allow you to obtain higher privileges only when you need them, similar to User Account Control (UAC) introduced in Windows Vista
- su is used to fully switch to another user account (typically root)
  - The "-" option lets you inherit the user's full environment (this is recommended in most cases)
  - If you specify a username, you will switch to that user; if you don't specify a username, it switches to root by default
  - You need to provide the password of the account you're switching to (unless you're root)
  - The "-c" option can be used to run a command instead of obtaining a shell
  - On some systems, you have to be in the "wheel" group to switch to the root account

Su and Sudo

Linux also offers tools that allow administrative tasks to be performed from user accounts without granting the account unnecessary permissions. Similar to User Account Control (UAC) introduced in Windows Vista, these two tools are named "su" and "sudo" and have been used by system administrators for many years.

The "su" (substitute user) command is used to fully switch to another user account, typically root. When using su, you are prompted for the password of the account you are switching to (unless you are root, in which case you can switch to any account with a valid login shell). The "-" option can be used to run the new shell as a login shell, inheriting the full environment (this is recommended in most cases). To switch to root, you can just run **su** or **su -**.

```
jdoe@localhost$ su -
Password: <enter root password>
root@localhost$
```

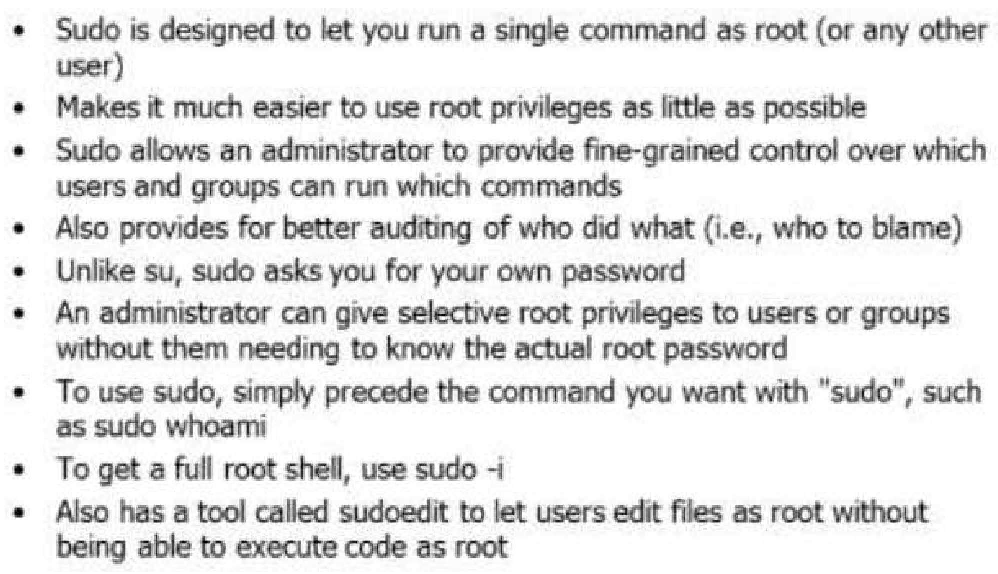
To switch to a specific user, just specify the username.

```
jdoe@localhost$ su - testuser
Password: <enter testuser's password>
testuser@localhost$
```

To run a command instead of getting a shell, use the "-c" option.

```
jdoe@localhost$ su -c whoami
Password: <enter root's password>
root
jdoe@localhost$
```

Some Linux systems require you to be in the "wheel" group in order to switch to the root account using su, even if you know the correct password.



- Sudo is designed to let you run a single command as root (or any other user)
- Makes it much easier to use root privileges as little as possible
- Sudo allows an administrator to provide fine-grained control over which users and groups can run which commands
- Also provides for better auditing of who did what (i.e., who to blame)
- Unlike su, sudo asks you for your own password
- An administrator can give selective root privileges to users or groups without them needing to know the actual root password
- To use sudo, simply precede the command you want with "sudo", such as sudo whoami
- To get a full root shell, use sudo -i
- Also has a tool called sudoedit to let users edit files as root without being able to execute code as root

Sudo

The "sudo" command is designed to let a user run a single command as root (or any other user), rather than having to completely switch to a full shell as the other user. This makes it much easier to avoid using the root account except when absolutely necessary. Sudo allows an administrator to specify specific users and groups that have permission to run commands as root, and also allows the administrator to specify specific commands that each user or group can run.

Unlike su, sudo asks you for your own password. This allows administrators to grant root privileges to users without them needing to know the actual root password, which also allows for privileges to be easily audited and revoked. To use sudo, simply put the command "sudo" before the command that you want to run as root. For example, to run the "whoami" command as root, run:

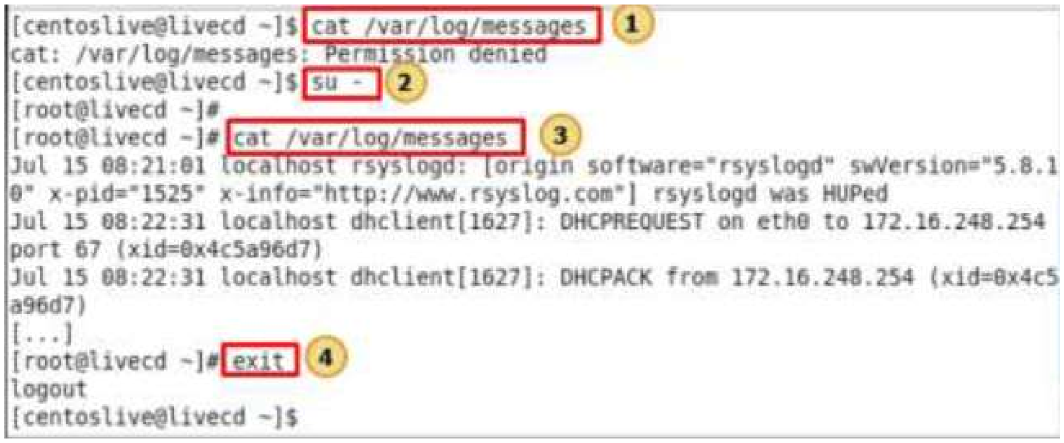
```
jdoe@localhost$ sudo whoami
[sudo] password for jdoe: <enter jdoe's password>
root
jdoe@localhost$
```

If you still want to get a full root shell isntead of just running a single command, you can use the -i option.

```
jdoe@localhost$ sudo -i
[sudo] password for jdoe: <enter jdoe's password>
root@localhost$
```

Sudo also has a tool called "sudoedit" that allows you to edit files with root privileges without having full code execution.

```
jdoe@localhost$ sudoedit /etc/shadow
```



```
[centoslive@livecd ~]$ cat /var/log/messages
cat: /var/log/messages: Permission denied
[centoslive@livecd ~]$ su -
[root@livecd ~]#
[root@livecd ~]# cat /var/log/messages
Jul 15 08:21:01 localhost rsyslogd: [origin software="rsyslogd" swVersion="5.8.1
0" x-pid="1525" x-info="http://www.rsyslog.com"] rsyslogd was HUPed
Jul 15 08:22:31 localhost dhclient[1627]: DHCPREQUEST on eth0 to 172.16.248.254
port 67 (xid=0x4c5a96d7)
Jul 15 08:22:31 localhost dhclient[1627]: DHCPACK from 172.16.248.254 (xid=0x4c5
a96d7)
[...]
[root@livecd ~]# exit
logout
[centoslive@livecd ~]$
```

Exercise: Users and Groups

Let's practice using our newfound knowledge of users, su, and sudo and walkthrough these steps:

1. Notice that your prompt says "centoslive@livecd". This means you're logged into an account called "centoslive" on the machine called "livecd". The user account "centoslive" has limited privileges. Let's demonstrate that by trying to read a protected file with the following command:

   ```
   $ cat /var/log/messages
   ```

   You should receive a "Permission Denied" error, because the user "centoslive" doesn't have permissions to read the /var/log/messages file.

2. Let's "switch user" with the "su" command:

   ```
   $ su -
   ```

   Normally, you would be prompted for root's password here, but since this is a special LiveCD the root password has been deleted (this can be accomplished with **`passwd -d root`**). Notice that your prompt now says "root@livecd" and the dollar sign ($) has changed to a pound sign (#). This means that you're now logged into the "root" account and you have full privileges in this VM.

3. Now, let's try reading the protected file again:

   ```
   # cat /var/log/messages
   ```

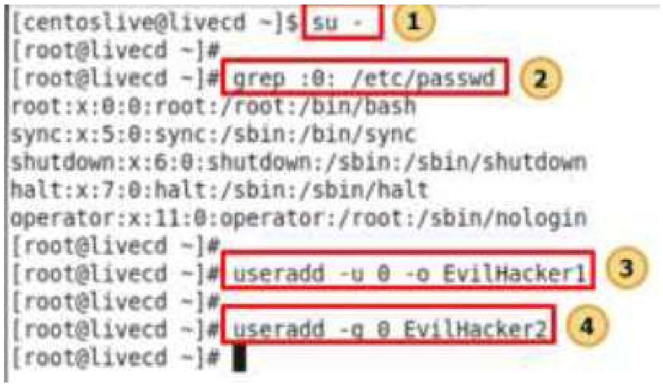
   The command should now succeed and you should see the contents of the /var/log/messages file. This is because the root user can read ANY file on the system. Your output will be different that that shown above.

4. It's good security practice to only use the root account when absolutely necessary. So let's exit the root account and return to the centos account by typing:

   ```
   # exit
   ```

   Alternatively, you could use the keyboard shortcut Ctrl-D.

   Your terminal prompt should change from the pound sign (#) to a dollar sign ($). This signifies that you are



```
[centoslive@livecd ~]$ su -
[root@livecd ~]#
[root@livecd ~]# grep :0: /etc/passwd
root:x:0:0:root:/root:/bin/bash
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
operator:x:11:0:operator:/root:/sbin/nologin
[root@livecd ~]#
[root@livecd ~]# useradd -u 0 -o EvilHacker1
[root@livecd ~]#
[root@livecd ~]# useradd -g 0 EvilHacker2
[root@livecd ~]#
```

Exercise: Finding Accounts with Root Privileges (1)

As you will recall, users with a UID of 0 have full access to the system. This access is often referred to as root level access. Let's walkthrough and exercise to create then find accounts with UID 0 or GID 0.

1. In your CentOS VM, open a terminal and type "su -" to become root.

   ```
   $ su -
   ```

2. Type the command and note the output:

   ```
   # grep :0: /etc/passwd
   ```

   There should be 5 accounts that have a 0 in the UID or GID fields. The "grep" command will filter for lines matching ":0:" in the /etc/passwd file.

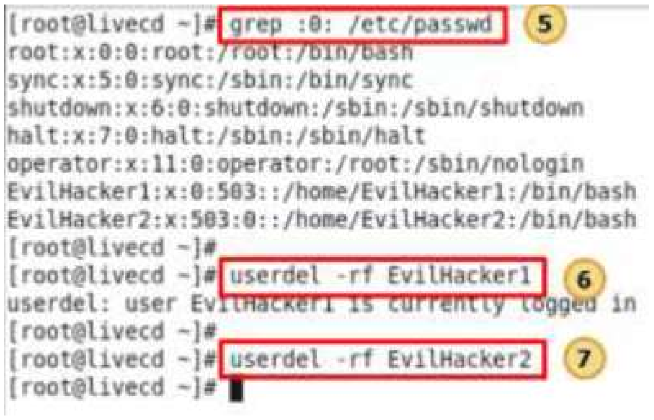
3. Add a new user with UID 0 with the following command:

   ```
   # useradd -u 0 -o EvilHacker1
   ```

   The -u option specifies the UID value, otherwise the OS will pick the next available number. The -o option will allow us to create an account with a non-unique UID (the existing root account has UID 0).

4. Add a new user with GID 0 with the following command:

   ```
   # useradd -g 0 EvilHacker2
   ```



```
[root@livecd ~]# grep :0: /etc/passwd
root:x:0:0:root:/root:/bin/bash
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
operator:x:11:0:operator:/root:/sbin/nologin
EvilHacker1:x:0:503::/home/EvilHacker1:/bin/bash
EvilHacker2:x:503:0::/home/EvilHacker2:/bin/bash
[root@livecd ~]#
[root@livecd ~]# userdel -rf EvilHacker1
userdel: user EvilHacker1 is currently logged in
[root@livecd ~]#
[root@livecd ~]# userdel -rf EvilHacker2
[root@livecd ~]#
```

Exercise: Finding Accounts with Root Privileges (2)

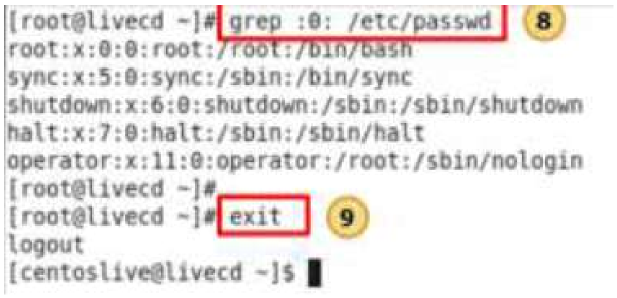
5. Run the grep command again. You could type the command all over again, or you could simply press the Up arrow three times to retrieve that command from your command history and then press Enter. There should now be 7 accounts shown. Notice that the EvilHacker1 account has a 0 in the UID field and EvilHacker2 has a 0 in the GID field.
6. Delete the EvilHacker account with the following command:

```
# userdel -rf EvilHacker
```

   You will see a warning that "user EvilHacker1 is currently logged in" because there are processes running as the root user (also UID 0). You can ignore this error as EvilHacker1 has been deleted. If you want to confirm you could run the previous grep command again.
7. Delete the EvilHacker2 account by pressing the Up arrow, replacing the 1 with a 2, and pressing Enter:

```
# userdel -rf EvilHacker2
```

   At this point both of the accounts have been deleted.



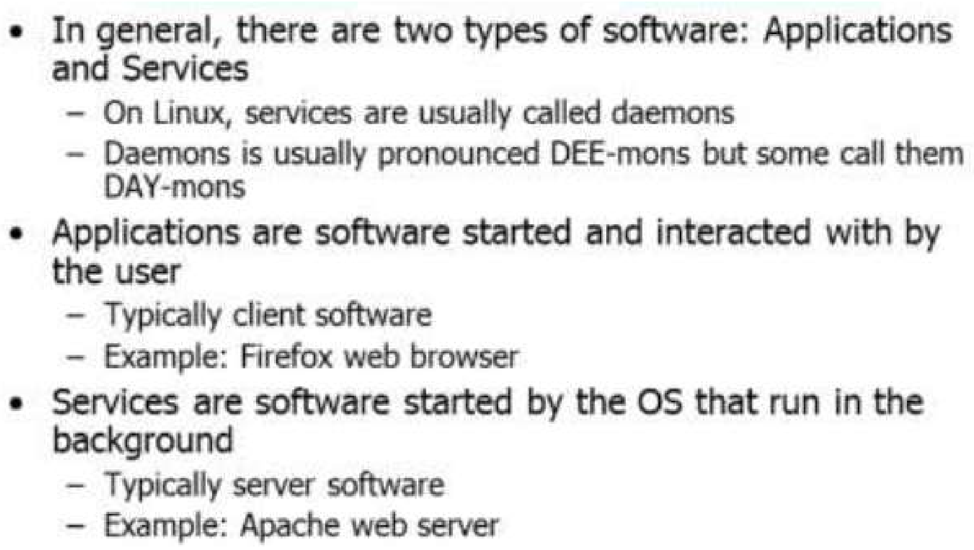
```
[root@livecd ~]# grep :0: /etc/passwd
root:x:0:0:root:/root:/bin/bash
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
operator:x:11:0:operator:/root:/sbin/nologin
[root@livecd ~]#
[root@livecd ~]# exit
logout
[centoslive@livecd ~]$
```

Exercise: Finding Accounts with Root Privileges (3)

8. Press the Up arrow three times to retrieve the grep command again and press Enter. You should now be back to 5 accounts with a 0 in the UID or GID fields. The two extra accounts, EvilHacker1 and EvilHacker2, have been deleted.

9. Type "exit" to leave root level access.

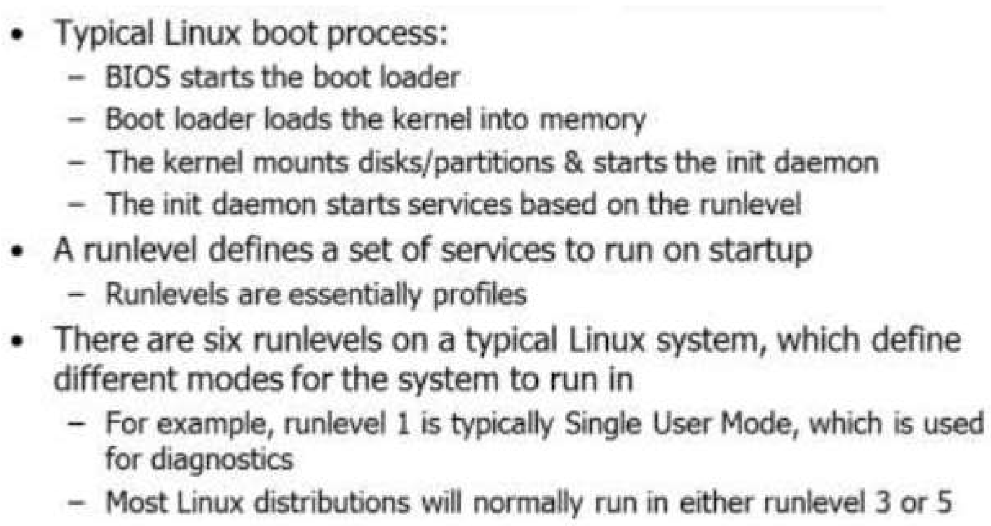
The grep command used here is useful to find accounts that have a UID or GID of 0. With the right access, an attacker could create a backdoor account that would give him full access to the system. This is a good way to see if any such accounts exist on the system. This command is referenced in the SANS Linux Intrusion Discovery Cheat Sheet, located at: http://www.sans.org/security-resources/linsacheatsheet.pdf



- In general, there are two types of software: Applications and Services
  - On Linux, services are usually called daemons
  - Daemons is usually pronounced DEE-mons but some call them DAY-mons
- Applications are software started and interacted with by the user
  - Typically client software
  - Example: Firefox web browser
- Services are software started by the OS that run in the background
  - Typically server software
  - Example: Apache web server

Applications and Services

There are two categories of software that run on top of the operating system: applications and services (commonly called daemons on Linux). Applications are started by the user, interacted with, and then closed. For example, you sit down at your Linux VM, open your Firefox web browser, search for something on google.com, and then close your browser down. That's an application. Services, on the other hand, typically start when the computer boots up and run until the computer shuts down. An example of a service would be the HTTP service running on one of Google's servers that your web browser connects to. That HTTP service is started automatically by the operating system and will continue running until the operating system shuts down or an administrative user stops the service.



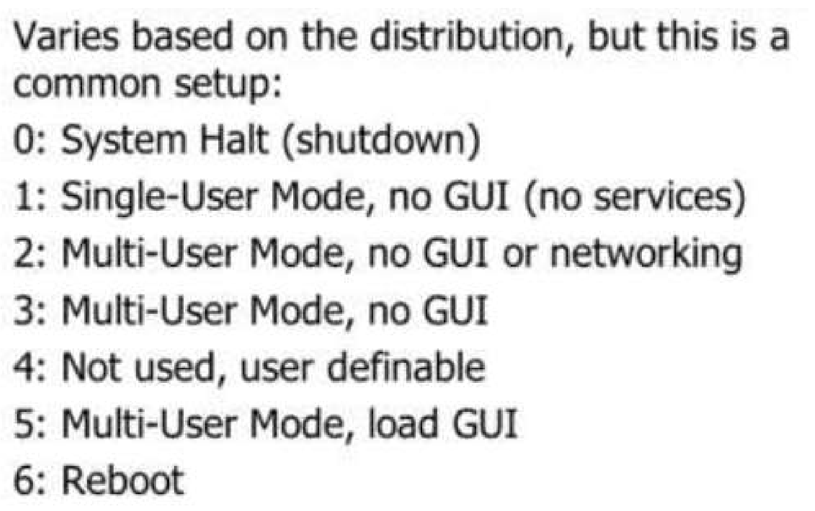
- Typical Linux boot process:
  - BIOS starts the boot loader
  - Boot loader loads the kernel into memory
  - The kernel mounts disks/partitions & starts the init daemon
  - The init daemon starts services based on the runlevel
- A runlevel defines a set of services to run on startup
  - Runlevels are essentially profiles
- There are six runlevels on a typical Linux system, which define different modes for the system to run in
  - For example, runlevel 1 is typically Single User Mode, which is used for diagnostics
  - Most Linux distributions will normally run in either runlevel 3 or 5

Linux Boot Process

In a traditional Linux system, the boot process goes like this:

- The computer BIOS starts the boot loader, a program on the boot device (such as a hard drive or DVD)
- The boot loader loads the kernel into memory
- The kernel mounts the disk partitions needed to run the system and starts the init program, which is the first user-mode program on the machine
- Init starts services based on the runlevel

In the last step of the boot process, we said that init starts services based on the runlevel. What's a runlevel? A runlevel is a way of telling the operating system what you want to do with it so that it can start services appropriately. Most of the time, the runlevel will be 5, which tells Linux to start all normal services and the graphical user interface (GUI). An alternative would be runlevel 3, which is similar to runlevel 5, but without the GUI. However, for maintenance or emergency situations, we may choose to go to runlevel 1. This is a text-only mode where just the root user can login and no system services are started. Other runlevels usually have different sets of services that are started automatically.



Varies based on the distribution, but this is a common setup:

0: System Halt (shutdown)

1: Single-User Mode, no GUI (no services)

2: Multi-User Mode, no GUI or networking

3: Multi-User Mode, no GUI

4: Not used, user definable

5: Multi-User Mode, load GUI

6: Reboot

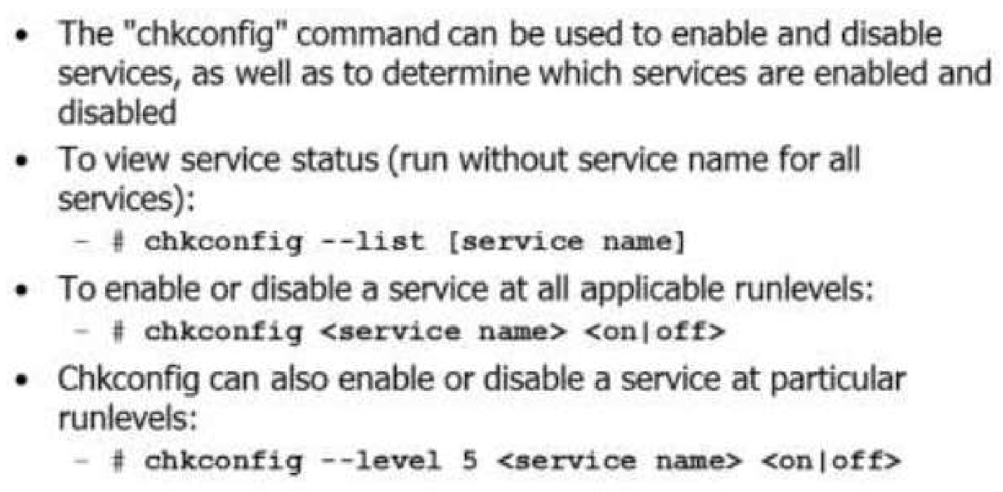
CentOS Runlevels

Most Linux system use a set of runlevels similar to the following (though some Linux distributions define runlevels 2-5 differently):

- 0: Shuts down all services and the operating system itself gracefully.
- 1: Enters Single-User Mode, which is typically an emergency rescue mode that simply loads a command shell as the root account (and no other services)
- 2: Boots into Multi-User Mode, but does not load any services related to networking or a windowing system (GUI)
- 3: Boots into Multi-User Mode with networking, but does not load any services related to a windowing system (GUI)
- 4: Not typically used for anything
- 5: Boots into Multi-User Mode, and loads the windowing system (GUI)
- 6: Shuts down all services and reboots the operating system gracefully

For additional information on this subject check out the link below:

http://www.techotopia.com/index.php/Configuring_CentOS_6_Runlevels_and_Services



- The "chkconfig" command can be used to enable and disable services, as well as to determine which services are enabled and disabled
- To view service status (run without service name for all services):
  - `# chkconfig --list [service name]`
- To enable or disable a service at all applicable runlevels:
  - `# chkconfig <service name> <on|off>`
- Chkconfig can also enable or disable a service at particular runlevels:
  - `# chkconfig --level 5 <service name> <on|off>`

chkconfig

On Red Hat-based systems, the "chkconfig" command can be used to enable and disables services, as well as to determine which services are enabled and disabled at all runlevels.

To view the status of all services at all runlevels, run:

```
# chkconfig -list
```

To view the status of a particular service, run:

```
# chkconfig --list [service name]
```

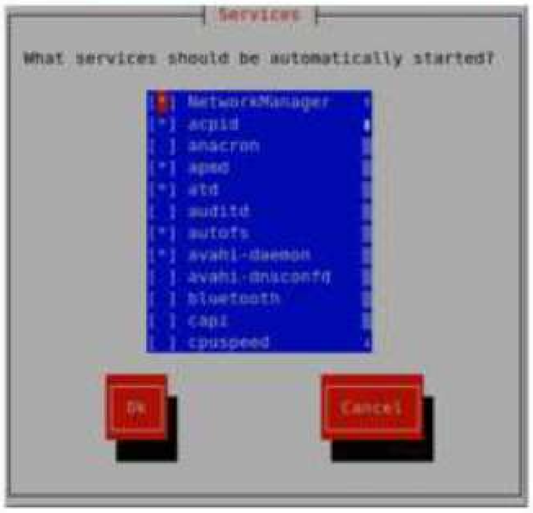
To enable or disable a service (e.g. SSH) at all applicable runlevels, run:

```
# chkconfig sshd off
```

Chkconfig can also enable or disable a service at particular runlevels using the "--level" option. Note that it is possible to specify more than one runlevel at once by simply combining the numbers. For example, to enable to SSHD service at runlevels 3 & 5, run:

```
# chkconfig --level 35 sshd on
```

- The command line tool "ntsysv" provides a semi-graphical interface for managing services at a particular runlevel
- Use the arrow keys to select a service, then use the space bar to enable or disable it



```
Services
What services should be automatically started?
[*] NetworkManager
[*] acpid
[ ] anacron
[*] apmd
[*] atd
[ ] auditd
[*] autofs
[*] avahi-daemon
[ ] avahi-dnsconfd
[ ] bluetooth
[ ] capi
[ ] cpuspeed
```

ntsysv

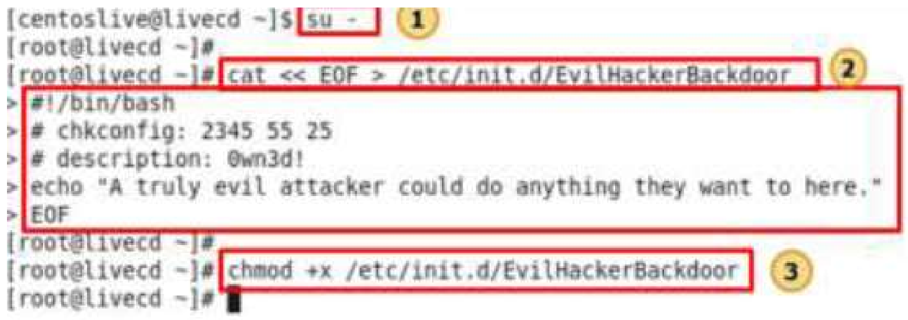
The "ntsysv" tool runs at the commandline, but provides an ncurses-based interface for managing services at a particular runlevel (ncurses is a library for creating semi-graphical applications at the CLI). By default, it edits the current runlevel, but it also supports the same "--levels" syntax as chkconfig to edit one or more arbitrary runlevels at a time.

To manage services, simply use the arrow keys to select a service, and use the space bar to enable or disable it. Then, press the tab key to move the cursor to the "Ok" button and press the space bar to "click" it.

- One technique attackers use is to create a backdoor in /etc/rc.d/init.d/
- By creating their own service and configuring it at the default runlevel, the attacker's service will start every time the machine boots
- You can use the tools in the previous section (such as chkconfig) to look for suspicious services

Init Backdoor

One technique used by computer attackers is to create a backdoor in /etc/rc.d/init.d/. Recall that this is the directory that specifies system services. If attackers can install their own service in this directory (by writing a simple script) and create an entry in the appropriate rc.d directory (such as rc5.d if we're in runlevel 5), then the attackers' service will start every time the machine boots and the bad guys will always have access to the box. You can use the tools listed in the previous section to find services that look suspicious.



```
[centoslive@livecd ~]$ su -
[root@livecd ~]#
[root@livecd ~]# cat << EOF > /etc/init.d/EvilHackerBackdoor
> #!/bin/bash
> # chkconfig: 2345 55 25
> # description: Own3d!
> echo "A truly evil attacker could do anything they want to here."
> EOF
[root@livecd ~]#
[root@livecd ~]# chmod +x /etc/init.d/EvilHackerBackdoor
[root@livecd ~]#
```

Init Backdoors Exercise (1)

1. In your CentOS VM, start a terminal (using either the panel or desktop icon you created earlier) and become root using the following command:

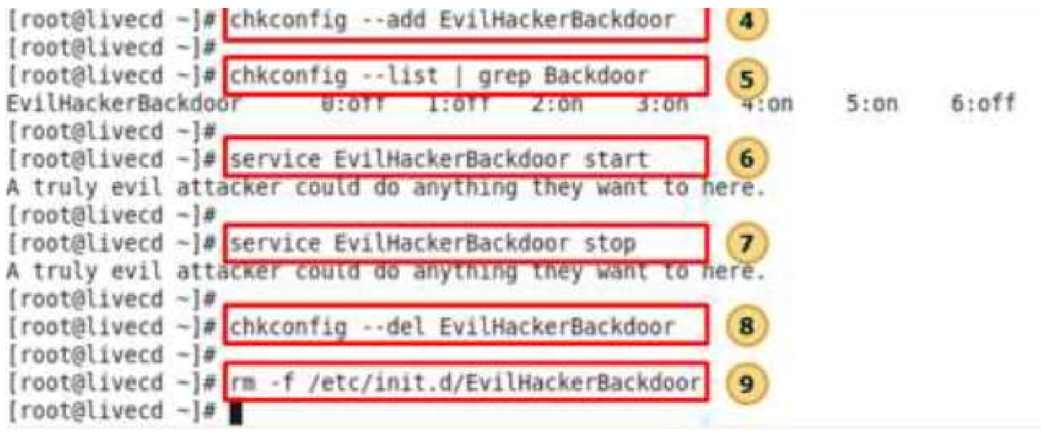
   ```
   $ su -
   ```

2. Now that you are root, type the following command to create a backdoor in /etc/init.d. Remember that the first pound sign (#) you see is the root prompt and you shouldn't type that. The pound signs on lines 2-4 are part of the EvilHackerBackdoor script and you SHOULD type those (they are important to the script). Type everything below to create the script. This techniques uses a "heredoc" to create the script.

   ```
   cat << EOF >> /etc/init.d/EvilHackerBackdoor
   #!/bin/bash
   # chkconfig: 2345 55 25
   # description: 0wn3d!
   echo "A truly evil attacker could do anything they want to here."
   EOF
   ```

3. Make the backdoor executable:

   ```
   # chmod +x /etc/init.d/EvilHackerBackdoor
   ```



```
[root@livecd ~]# chkconfig --add EvilHackerBackdoor        4
[root@livecd ~]#
[root@livecd ~]# chkconfig --list | grep Backdoor          5
EvilHackerBackdoor      0:off   1:off   2:on    3:on    4:on    5:on    6:off
[root@livecd ~]#
[root@livecd ~]# service EvilHackerBackdoor start          6
A truly evil attacker could do anything they want to here.
[root@livecd ~]#
[root@livecd ~]# service EvilHackerBackdoor stop           7
A truly evil attacker could do anything they want to here.
[root@livecd ~]#
[root@livecd ~]# chkconfig --del EvilHackerBackdoor        8
[root@livecd ~]#
[root@livecd ~]# rm -f /etc/init.d/EvilHackerBackdoor      9
[root@livecd ~]#
```

Init Backdoors Exercise (2)

4. Add the backdoor to all runlevels:

```
# chkconfig --add EvilHackerBackdoor
```

5. Verify that the backdoor was added (grep is used to search for a particular string of text):

```
# chkconfig --list | grep Backdoor
```

6. Start the backdoor:

```
# service EvilHackerBackdoor start
```

You should see the following text:

```
A truly evil attacker could do anything they want to here.
```

7. Stop the backdoor:

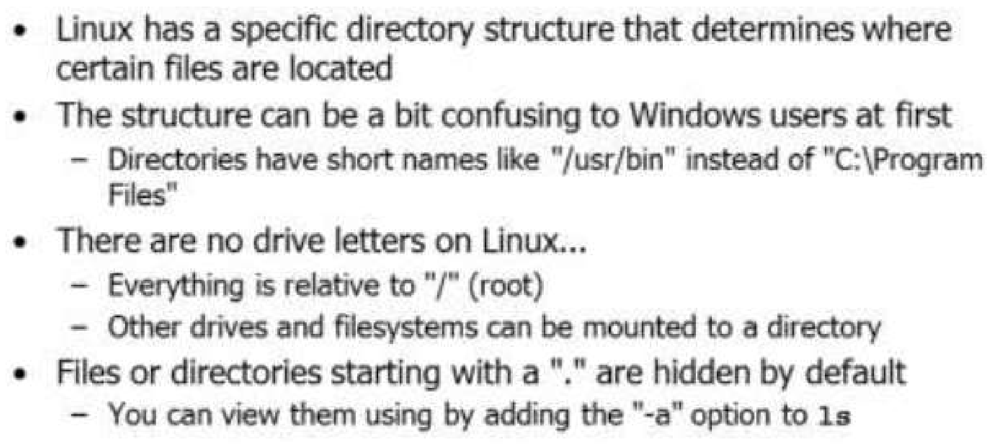
```
# service EvilHackerBackdoor stop
```

8. Remove the backdoor from all runlevels:

```
# chkconfig --del EvilHackerBackdoor
```

9. Remove the backdoor script:

```
# rm -f /etc/init.d/EvilHackerBackdoor
```

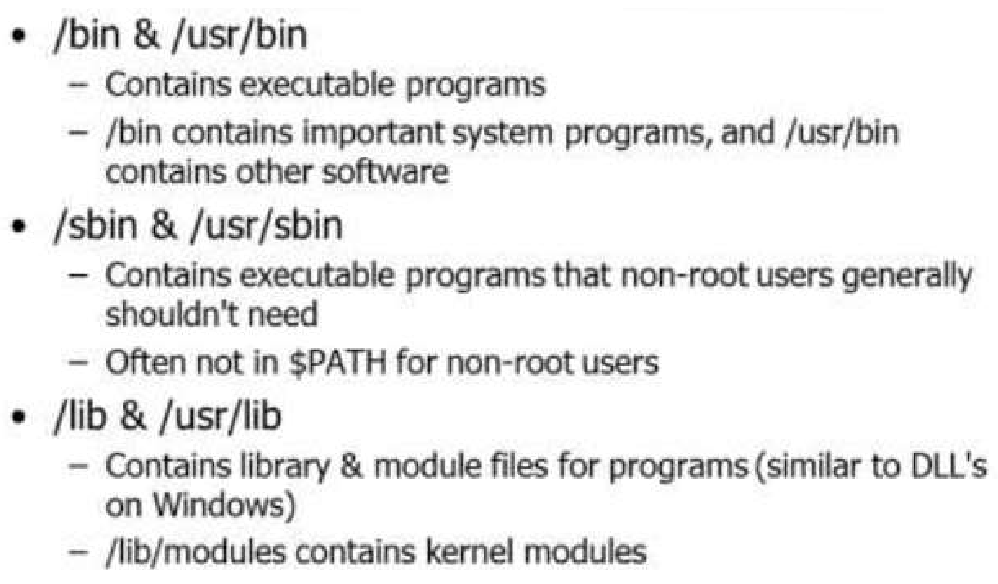


- Linux has a specific directory structure that determines where certain files are located
- The structure can be a bit confusing to Windows users at first
  - Directories have short names like "/usr/bin" instead of "C:\Program Files"
- There are no drive letters on Linux...
  - Everything is relative to "/" (root)
  - Other drives and filesystems can be mounted to a directory
- Files or directories starting with a "." are hidden by default
  - You can view them using by adding the "-a" option to `ls`

Directory Structure

Linux, like Windows, has specific directories for users to store their files and other directories for the Operating System to store its files. There are directories for the Operating System to store executable programs and separate directories for it to store libraries (Linux libraries are similar to Windows DLLs). Unlike Windows, every aspect of the Linux OS is represented as a file. The operating system's memory, input/output devices, and network adapters are all represented as files in the file system.

Files or directories starting with a dot (".") are hidden by default, similar to the "hidden" file attribute on Windows. You can view them using the "-a" option to `ls`.



- /bin & /usr/bin
  - Contains executable programs
  - /bin contains important system programs, and /usr/bin contains other software
- /sbin & /usr/sbin
  - Contains executable programs that non-root users generally shouldn't need
  - Often not in $PATH for non-root users
- /lib & /usr/lib
  - Contains library & module files for programs (similar to DLL's on Windows)
  - /lib/modules contains kernel modules

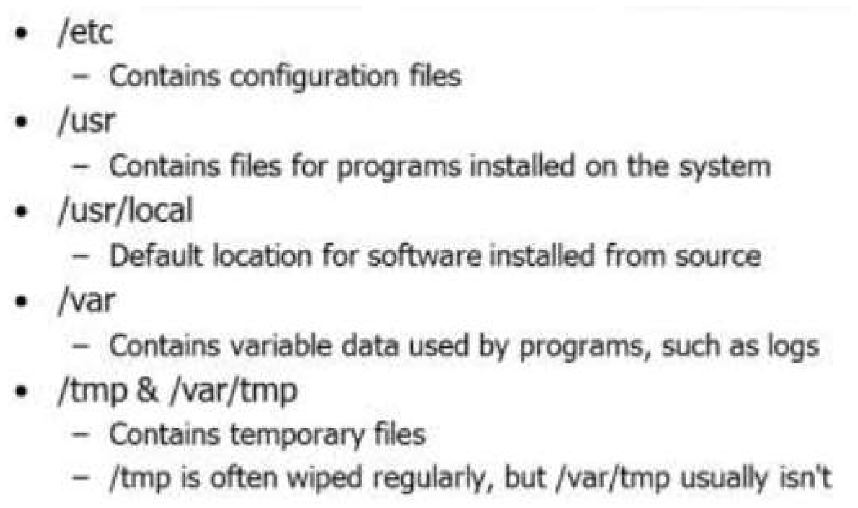
Important Directories (1)

Here are some of the most important directory locations on a Linux system:

/bin & /usr/bin: These contain executable programs. /bin contains important system programs (such as cp, ls, and mv), while /usr/bin contains other software installed on the system.

/sbin & /usr/sbin: These contain executable programs that non-root users generally shouldn't need, such as "ifconfig" for network interface configuration. Depending on the distribution, these directories are sometimes not in a regular user's $PATH.

/lib & /usr/lib: These contain library and module files for programs, which are similar to DLL files on Windows. /lib/modules contains kernel modules.



- /etc
  - Contains configuration files
- /usr
  - Contains files for programs installed on the system
- /usr/local
  - Default location for software installed from source
- /var
  - Contains variable data used by programs, such as logs
- /tmp & /var/tmp
  - Contains temporary files
  - /tmp is often wiped regularly, but /var/tmp usually isn't

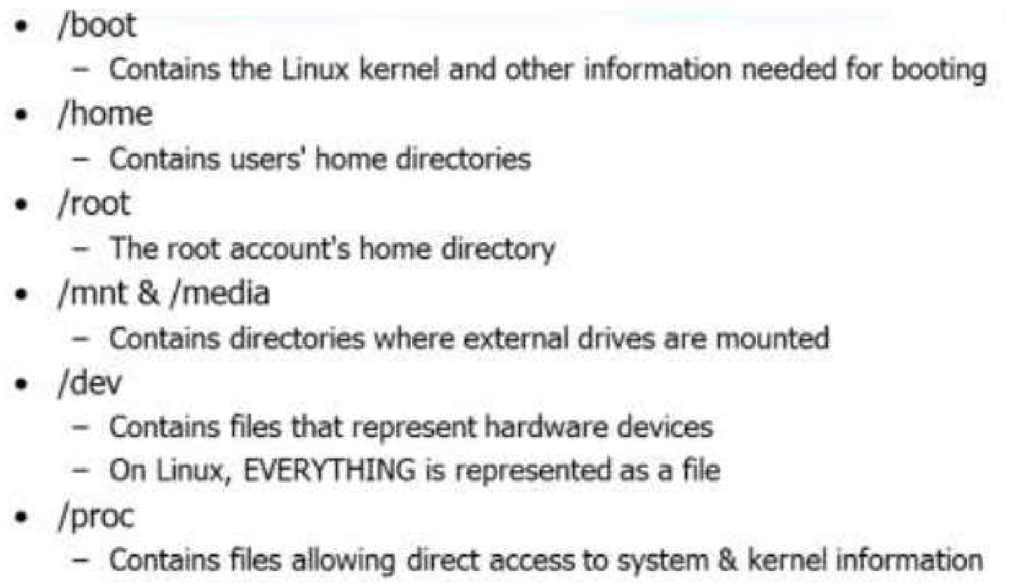
/etc: Contains configuration files for the system and for other software installed on the system. For example, /etc/resolv.conf contains the DNS servers the system will use to resolve names, and /etc/hosts contains local name to IP address mappings.

/usr: Contains files for programs installed on the system. This has a set of subdirectories that matches the rest of the system, such as bin, sbin, and lib.

/usr/local: The default location for software installed from source. Like /usr, this also contains a set of subdirectories including bin, sbin, and lib.

/var: Contains variable data used by programs, such as logs. The default web root is often /var/www/.

/tmp & /var/tmp: These contain temporary files used by programs or the system. /tmp is usually cleaned out automatically either during reboot or on a regular basis, while /var/tmp usually isn't.



- /boot
  - Contains the Linux kernel and other information needed for booting
- /home
  - Contains users' home directories
- /root
  - The root account's home directory
- /mnt & /media
  - Contains directories where external drives are mounted
- /dev
  - Contains files that represent hardware devices
  - On Linux, EVERYTHING is represented as a file
- /proc
  - Contains files allowing direct access to system & kernel information

/boot: Contains the Linux kernel (the core of the operating system) and other information needed for booting.

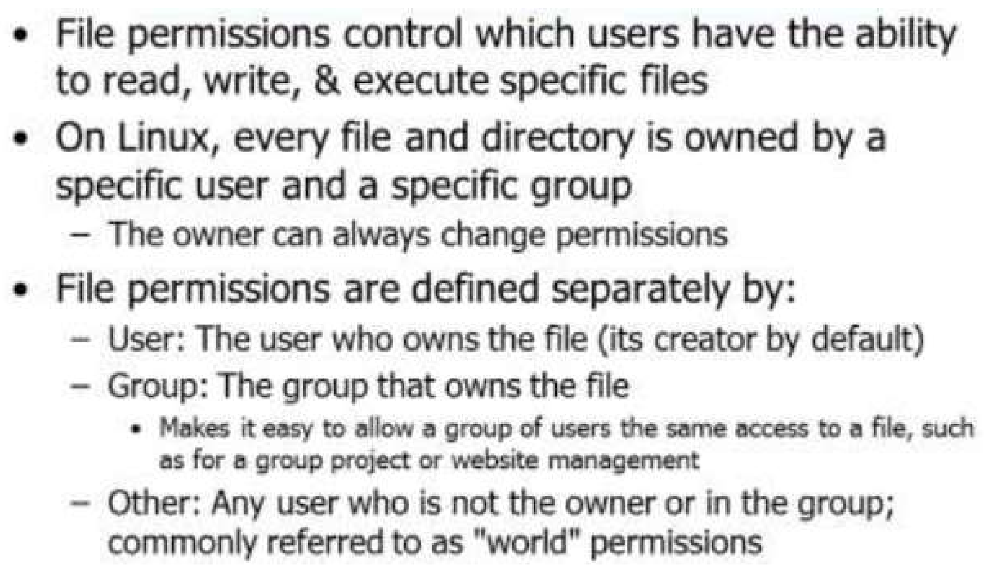
/home: Contains users' home directories (containing their personal files, settings, etc.).

/root: The root account's home directory.

/mnt & /media: These contain directories where external & network drives are mounted. /mnt generally contains generic mount points like /mnt/floppy and /mnt/cdrom, while /media is usually handled by a daemon that creates new directories on the fly with the name of the device (such as /media/SANS_Network_Tools).

/dev: Contains files that represent hardware devices (on Linux, ALL devices are represented by a file).

/proc: Contains virtual files allowing direct access to system & kernel information. For example, there are numbered subdirectories that correspond to each running process on the system. The numbers correspond to the process ID.



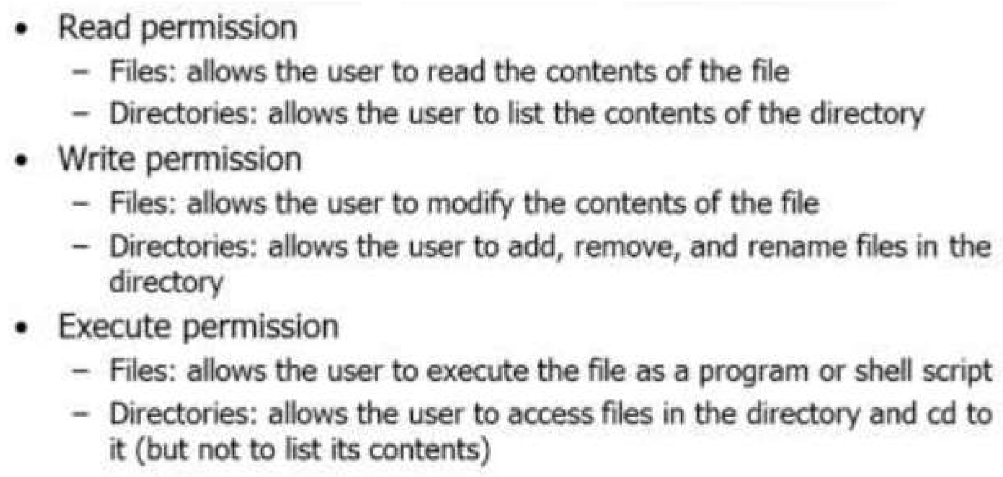
- File permissions control which users have the ability to read, write, & execute specific files
- On Linux, every file and directory is owned by a specific user and a specific group
  - The owner can always change permissions
- File permissions are defined separately by:
  - User: The user who owns the file (its creator by default)
  - Group: The group that owns the file
    - Makes it easy to allow a group of users the same access to a file, such as for a group project or website management
  - Other: Any user who is not the owner or in the group; commonly referred to as "world" permissions

File Ownership & Permissions

File permissions are used to control who can read, write and/or execute files on the hard drive. Ownership is used to specify who is considered the owner of a file. Permissions and ownership work together to control file system privileges.

On Linux, every file and directory is owned by a specific user and a specific group. The owner is always able to change the permissions, even if the current permissions don't allow the owner to access the file. This allows file permissions to be defined separately by the user who owns the file, the group that owns the file, and all other users on the system (commonly referred to as "world").

Users can change the group that owns any file they own (using "chgrp"), but only root can change the user that owns a file (using "chown").



- Read permission
  - Files: allows the user to read the contents of the file
  - Directories: allows the user to list the contents of the directory
- Write permission
  - Files: allows the user to modify the contents of the file
  - Directories: allows the user to add, remove, and rename files in the directory
- Execute permission
  - Files: allows the user to execute the file as a program or shell script
  - Directories: allows the user to access files in the directory and cd to it (but not to list its contents)

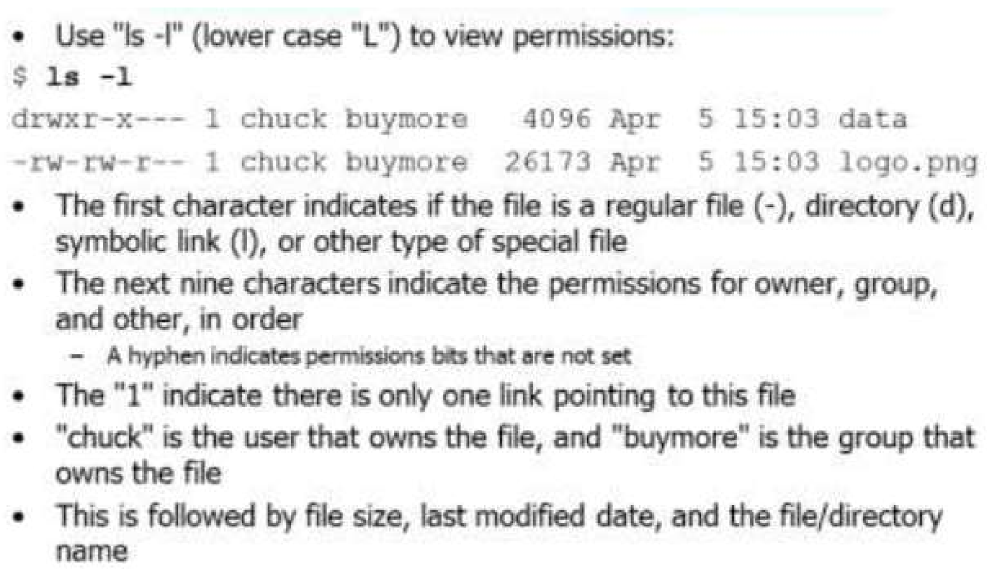
File Permissions

Linux has three types of permissions on a file: read, write, and execute.

Read permission allows the contents of files to be read, and the contents of directories to be listed.

Write permission allows the contents of a file to be modified, and allows users to add, remove, and rename files in a directory (note that even with write permission on a file, you cannot delete it without write permission on the directory!).

Execute permissions allows a file to be executed as a program or shell script, and allows users to enter and access files in a directory (but not list its contents).



- Use "ls -l" (lower case "L") to view permissions:

```
$ ls -l
drwxr-x--- 1 chuck buymore   4096 Apr  5 15:03 data
-rw-rw-r-- 1 chuck buymore  26173 Apr  5 15:03 logo.png
```

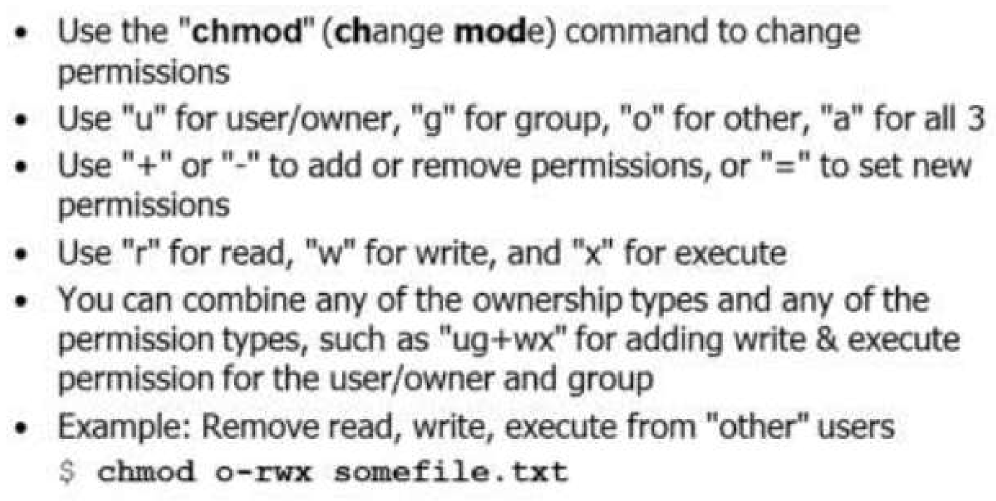
- The first character indicates if the file is a regular file (-), directory (d), symbolic link (l), or other type of special file
- The next nine characters indicate the permissions for owner, group, and other, in order
  - A hyphen indicates permissions bits that are not set
- The "1" indicate there is only one link pointing to this file
- "chuck" is the user that owns the file, and "buymore" is the group that owns the file
- This is followed by file size, last modified date, and the file/directory name

Viewing Permissions

You can view permissions on files and directories using the long listing "-l" (lowercase L) option to "ls". This will display a listing of the current or specified directory, one item per line.

The first character indicates if the file is a regular file (-), directory (d), symbolic link (lower-case L), or other type of special file. The next nine characters indicate the permissions for the file, in a particular order (rwx). The first 3 indicate read (r), write (w), and execute (x) for the user/owner, the next three for the group, and the last three for all other users. A hyphen indicates that the given permission is not set on that file or directory for the particular permissions group. The "1" (numeral one) after the permission bits indicates that there is only one hardlink to the file on the filesystem. "chuck" is the user that owns the file, and "buymore" is the group that owns the file. This is followed by the file size (usually 4096 for directories), the last modified date, and the file name.

In the example above, "data" is a directory owned by the user "chuck", and the group "buymore". The user "chuck" has read, write, and execute permissions, the group "buymore" has read and execute permissions, and all other users have no access at all. "logo.png" is a file: "chuck" has read and write permissions (but not execute), the "buymore" group has read and write permissions, and all other users have read permission only.



- Use the "**chmod**" (**ch**ange **mod**e) command to change permissions
- Use "u" for user/owner, "g" for group, "o" for other, "a" for all 3
- Use "+" or "-" to add or remove permissions, or "=" to set new permissions
- Use "r" for read, "w" for write, and "x" for execute
- You can combine any of the ownership types and any of the permission types, such as "ug+wx" for adding write & execute permission for the user/owner and group
- Example: Remove read, write, execute from "other" users

```
$ chmod o-rwx somefile.txt
```

Setting Permissions (Symbolic)

On Linux, the "chmod" command (short for "change mode") is used to change permissions on files and directories. There are two ways to specify the desired permissions: symbolic mode and octal/numeric mode. Symbolic mode is easiest when making changes to existing permissions (rather than setting all permission bits at once). Symbolic mode allows you to change permissions for one permission group (user, group, or other) at a time.

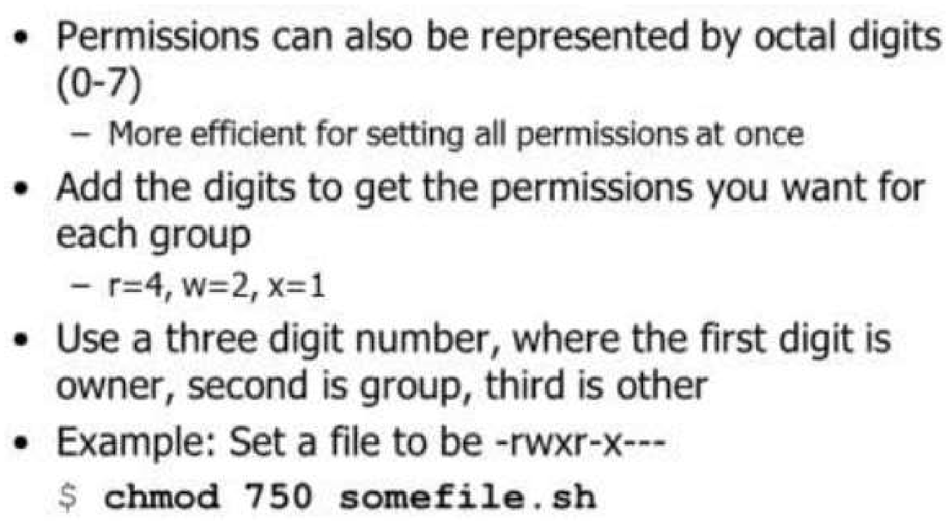
To set permissions in symbolic mode, first specify "u" for user/owner, "g" for group, "o" for other, or "a" for all three. Then, use a "+" to add permissions, a "-" to remove permissions, or "=" to assign new permissions. Then, use "r" for read, "w" for write, and "x" for execute. You can combine any of the ownership types and any of the permissions types in a single command, such as "ug+wx" for adding write & execute permission for the user/owner and group.

For example, to remove read, write, and execute permission from "other" users, run:

```
$ chmod o-rwx somefile.txt
```

To set a file to be read only (write and execute will be remove) for all users:

```
# chmod ugo=r somefile.txt
```

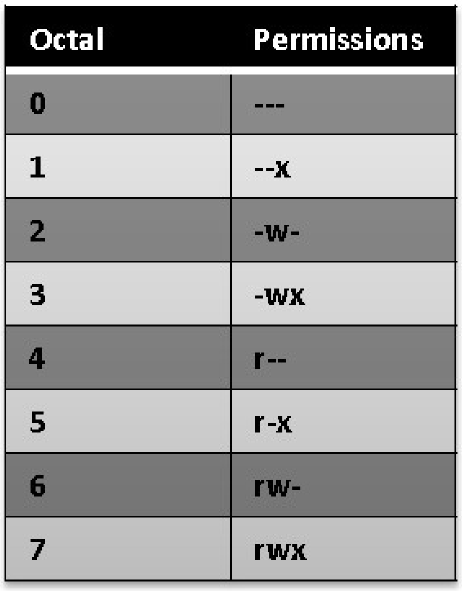


- Permissions can also be represented by octal digits (0-7)
  - More efficient for setting all permissions at once
- Add the digits to get the permissions you want for each group
  - r=4, w=2, x=1
- Use a three digit number, where the first digit is owner, second is group, third is other
- Example: Set a file to be -rwxr-x---

```
$ chmod 750 somefile.sh
```

Setting Permissions Octal

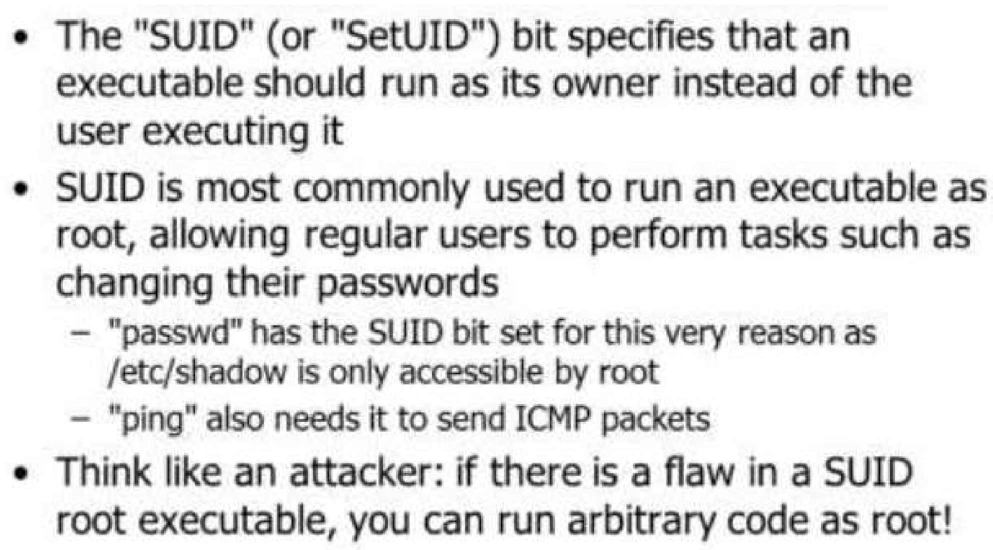
Linux permissions can also be represented by three octal digits (0-7), one for each permission group. This is a more efficient way to set all permissions bits at once. To use octal, add the permission bits together for each group, and then put them in order as a three digit number. Read permission is "4", write permission is "2", and execute permission is "1". All combinations can be expressed as a combination of these digits:



Octal	Permissions
0	---
1	--x
2	-w-
3	-wx
4	r--
5	r-x
6	rw-
7	rwx

For example, to set a file to have read, write, and execute permissions for the user/owner, read and execute permissions for the group, and no access for all other users (equivalent to symbolic -rwxr-x---), use the following command:

```
$ chmod 750 somefile.sh
```

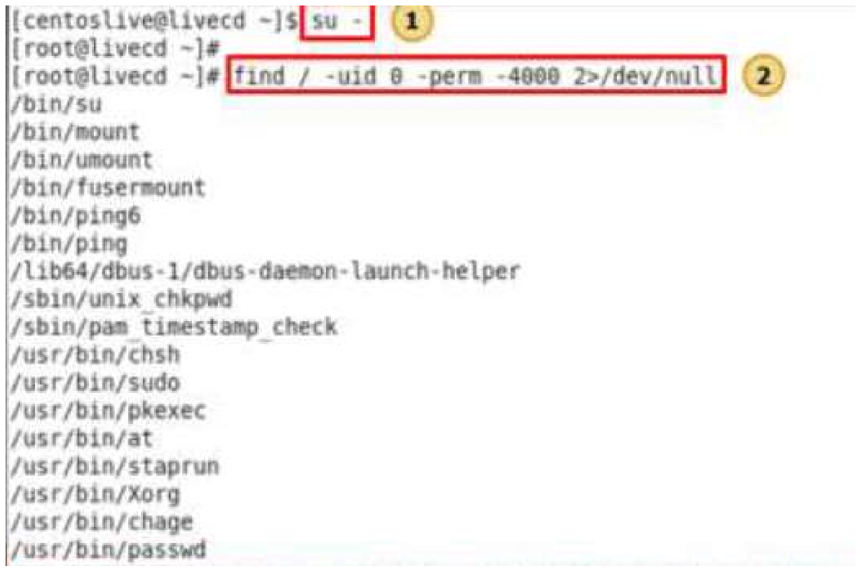


- The "SUID" (or "SetUID") bit specifies that an executable should run as its owner instead of the user executing it
- SUID is most commonly used to run an executable as root, allowing regular users to perform tasks such as changing their passwords
  - "passwd" has the SUID bit set for this very reason as /etc/shadow is only accessible by root
  - "ping" also needs it to send ICMP packets
- Think like an attacker: if there is a flaw in a SUID root executable, you can run arbitrary code as root!

SUID Bit

Some programs need to execute with the privileges of certain users. For example, the "passwd" program used to change your password has to be able to write your hashed password to the /etc/shadow file (which can only be written to by root). Therefore, the "passwd" program has the SUID bit set so that it will run as root, regardless of what account is actually executing it and the privileges they have.

Put your attacker hat on and think about how this could be abused. If a program is SUID root and it contains a flaw that can be exploited, then we could get full root access to the box! Because of this, programs that are SUID root have to very carefully written and audited to make sure they are free of any vulnerabilities.



```
[centoslive@livecd ~]$ su -
[root@livecd ~]#
[root@livecd ~]# find / -uid 0 -perm -4000 2>/dev/null
/bin/su
/bin/mount
/bin/umount
/bin/fusermount
/bin/ping6
/bin/ping
/lib64/dbus-1/dbus-daemon-launch-helper
/sbin/unix_chkpwd
/sbin/pam_timestamp_check
/usr/bin/chsh
/usr/bin/sudo
/usr/bin/pkexec
/usr/bin/at
/usr/bin/staprun
/usr/bin/Xorg
/usr/bin/chage
/usr/bin/passwd
```

SUID Bit Exercise

An attacker who has gained root access on your box may create a backdoor on your system that will enable him or her to easily recapture root access on your box. The simplest form of this backdoor is a shell with the SUID bit set. Search your computer to find all of the programs that are set to run with root privileges.

1. In your CentOS VM, open a terminal window and become root.

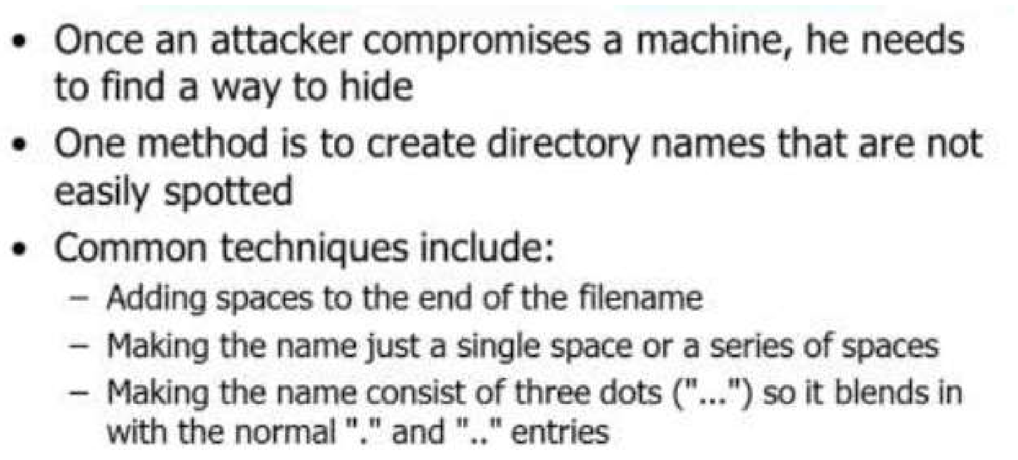
   ```
   $ su -
   ```

2. Search for all files that are SUID root using the following command from the SANS Linux Intrusion Discovery Cheat Sheet:

   ```
   # find / -uid 0 -perm -4000 2>/dev/null
   ```

   This command will search starting at the root of the file system (/), look for files owned by root (UID 0), and with the special SUID bit set (4000). We will throw away any error messages by sending STDERR (file descriptor 2) to /dev/null.

3. After a few seconds, the find command will return the list of files that are SUID root. Look through the list of files and think about why they must be SUID root. You should see familiar commands like "passwd", "su", "sudo", "ping", and other commands which might not be familiar. Use man to learn more about these unfamiliar commands and why they would need to be SUID root. For example, "man chfn".



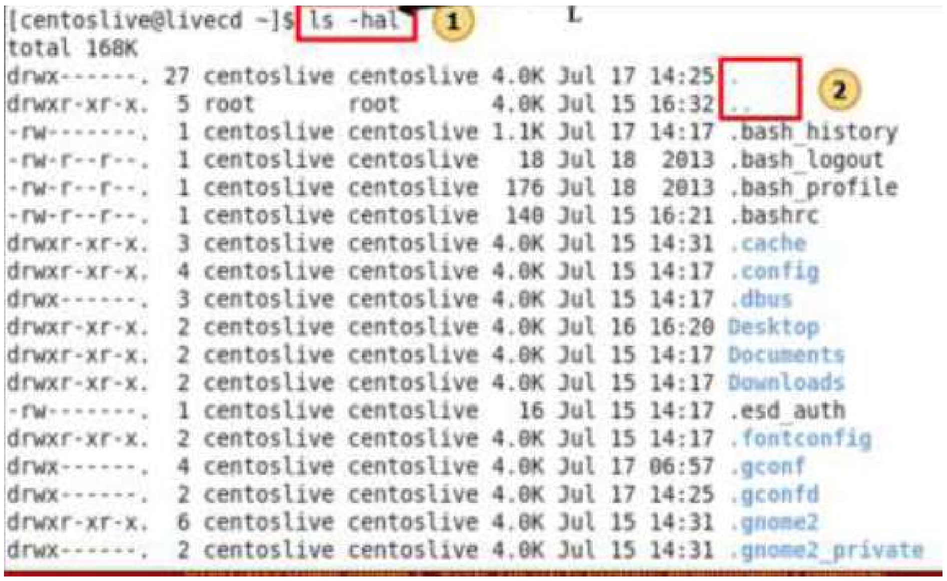
- Once an attacker compromises a machine, he needs to find a way to hide
- One method is to create directory names that are not easily spotted
- Common techniques include:
  - Adding spaces to the end of the filename
  - Making the name just a single space or a series of spaces
  - Making the name consist of three dots ("...") so it blends in with the normal "." and ".." entries

How Attackers Hide on the Linux Filesystem

Once an attacker has compromised a machine, one of his goals is to not be discovered. In order to do this, he needs to find ways to hide in the file system. One method of hiding is to create directory names that are not easily spotted. Common techniques used by attackers include:

- Adding spaces to the end of the filename
- Making the name just a single space or a series of spaces
- Making the name consist of three dots, so it blends in with the normal single-dot and double-dot directory entries

As a defender we need to aware of these techniques so we can find these hidden files and directories.



```
[centoslive@livecd ~]$ ls -hal
total 168K
drwx------. 27 centoslive centoslive 4.0K Jul 17 14:25 .
drwxr-xr-x.  5 root       root       4.0K Jul 15 16:32 ..
-rw-------.  1 centoslive centoslive 1.1K Jul 17 14:17 .bash_history
-rw-r--r--.  1 centoslive centoslive   18 Jul 18  2013 .bash_logout
-rw-r--r--.  1 centoslive centoslive  176 Jul 18  2013 .bash_profile
-rw-r--r--.  1 centoslive centoslive  140 Jul 15 16:21 .bashrc
drwxr-xr-x.  3 centoslive centoslive 4.0K Jul 15 14:31 .cache
drwxr-xr-x.  4 centoslive centoslive 4.0K Jul 15 14:17 .config
drwx------.  3 centoslive centoslive 4.0K Jul 15 14:17 .dbus
drwxr-xr-x.  2 centoslive centoslive 4.0K Jul 16 16:20 Desktop
drwxr-xr-x.  2 centoslive centoslive 4.0K Jul 15 14:17 Documents
drwxr-xr-x.  2 centoslive centoslive 4.0K Jul 15 14:17 Downloads
-rw-------.  1 centoslive centoslive   16 Jul 15 14:17 .esd_auth
drwxr-xr-x.  2 centoslive centoslive 4.0K Jul 15 14:17 .fontconfig
drwx------.  4 centoslive centoslive 4.0K Jul 17 06:57 .gconf
drwx------.  2 centoslive centoslive 4.0K Jul 17 14:25 .gconfd
drwxr-xr-x.  6 centoslive centoslive 4.0K Jul 15 14:31 .gnome2
drwx------.  2 centoslive centoslive 4.0K Jul 15 14:31 .gnome2_private
```

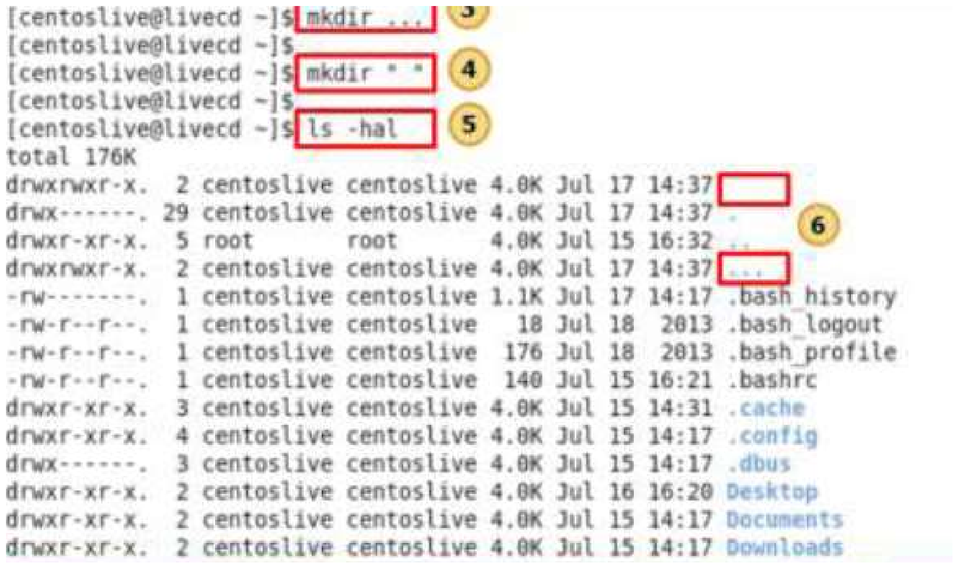
Exercise: Finding Hiden Directories (1)

1. In your CentOS VM, open a terminal window. We'll do the first part of this exercise as the "centos" user, so do not become root yet. Type the following command to show all the files in the current directory (/home/centos/):

   ```
   $ ls -hal
   ```

   Note: That is a lower case "L" at the end of the command

2. Notice that the first two entries are "." and "..". The single dot represents the current directory (/home/centoslive/). The double dot represents the parent directory (/home/). These entries are shown in every single directory in the file system, so a sysadmin would normally see them all the time.



```
[centoslive@livecd ~]$ mkdir ...
[centoslive@livecd ~]$
[centoslive@livecd ~]$ mkdir " "
[centoslive@livecd ~]$
[centoslive@livecd ~]$ ls -hal
total 176K
drwxrwxr-x.  2 centoslive centoslive 4.0K Jul 17 14:37
drwx------. 29 centoslive centoslive 4.0K Jul 17 14:37 .
drwxr-xr-x.  5 root       root       4.0K Jul 15 16:32 ..
drwxrwxr-x.  2 centoslive centoslive 4.0K Jul 17 14:37 ...
-rw-------.  1 centoslive centoslive 1.1K Jul 17 14:17 .bash_history
-rw-r--r--.  1 centoslive centoslive   18 Jul 18  2013 .bash_logout
-rw-r--r--.  1 centoslive centoslive  176 Jul 18  2013 .bash_profile
-rw-r--r--.  1 centoslive centoslive  140 Jul 15 16:21 .bashrc
drwxr-xr-x.  3 centoslive centoslive 4.0K Jul 15 14:31 .cache
drwxr-xr-x.  4 centoslive centoslive 4.0K Jul 15 14:17 .config
drwx------.  3 centoslive centoslive 4.0K Jul 15 14:17 .dbus
drwxr-xr-x.  2 centoslive centoslive 4.0K Jul 16 16:20 Desktop
drwxr-xr-x.  2 centoslive centoslive 4.0K Jul 15 14:17 Documents
drwxr-xr-x.  2 centoslive centoslive 4.0K Jul 15 14:17 Downloads
```

Exercise: Finding Hidden Directories (2)

3. Create a new directory called "..." with the following command:

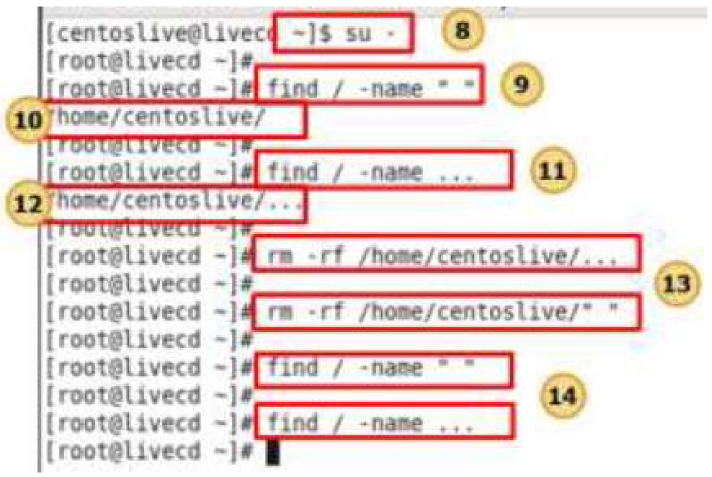
   ```
   $ mkdir ...
   ```

4. Create a new directory called " " (a single space) with the following command:

   ```
   $ mkdir " "
   ```

5. Press the Up arrow three times to retrieve the "ls -hal" command and press Enter.

6. Notice in the output how the new " " and "..." directories blend in with the normal "." and ".." entries. Think about how easy it would be for a sysadmin to overlook these stealthy directories. How would a sysadmin find them?



```
[centoslive@livecd ~]$ su -
[root@livecd ~]#
[root@livecd ~]# find / -name " "
/home/centoslive/
[root@livecd ~]#
[root@livecd ~]# find / -name ...
/home/centoslive/...
[root@livecd ~]#
[root@livecd ~]# rm -rf /home/centoslive/...
[root@livecd ~]#
[root@livecd ~]# rm -rf /home/centoslive/" "
[root@livecd ~]#
[root@livecd ~]# find / -name " "
[root@livecd ~]#
[root@livecd ~]# find / -name ...
[root@livecd ~]#
```

Exercise: Finding Hidden Directories (3)

8. Become root

```
$ su -
```

9. Use this command from the SANS Linux Cheat Sheet to look for filenames of a single space:

```
# find / -name " "
```

10. After a few seconds, the find command should display the location of the stealthy directory (/home/centoslive/). Note that there is a space after the final slash, you just can't tell that it's there.
11. Edit the previous command to look for the "..." directory. Press the Up arrow to retrieve the previous command, then press the backspace key twice and then type the following:

```
 ...
```

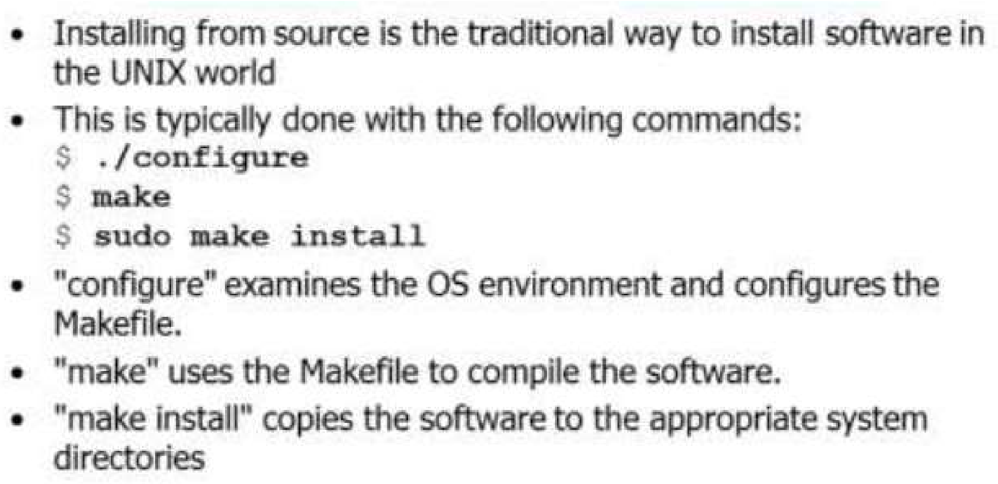
Now press Enter.

12. Again, the find command should display the location of the stealthy directory (/home/centoslive/...).
13. Remove the stealthy directories with the following commands:

```
# rm -rf /home/centos/...
# rm -rf /home/centos/" "
```

The "rm -rf" command recursively removes an entire directory structure even if the directories are not empty. Always be very careful when typing an "rm -rf" command, especially if you have root privileges! On a production system, one little typo could result in irreversible damage to the system. Since this is a LiveCD, if you accidentally wipe the entire file system, you can just reboot and everything will be back to normal.

14. Re-run the find commands above to verify that the stealthy directories have been removed.



- Installing from source is the traditional way to install software in the UNIX world
- This is typically done with the following commands:

```
$ ./configure
$ make
$ sudo make install
```

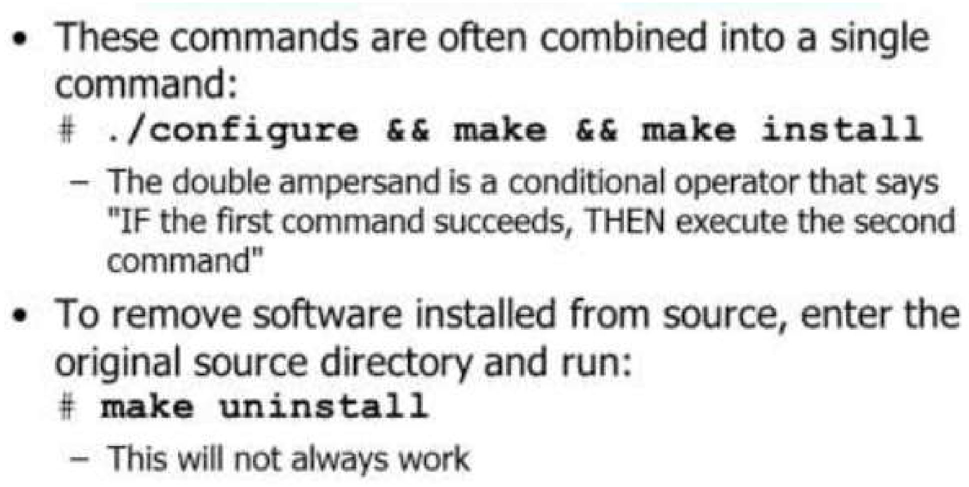
- "configure" examines the OS environment and configures the Makefile.
- "make" uses the Makefile to compile the software.
- "make install" copies the software to the appropriate system directories

Installing Software form Source (1)

The traditional method of software installation in the UNIX world is manually compiling source code into executable form. This is usually done with the following commands:

```
$ ./configure
$ make
$ sudo make install
```

The source tarball downloaded from the Internet contains a file called "configure", which we execute from the current directory by calling "./configure". This command examines the operating system and the software already installed on it and configures the Makefile which will be used in the next step. The "make" command uses the compiler that is already installed in the operating system to create binary executable programs. It references the Makefile created by the "./configure" step and compiles the source code into binary form. Finally, "make install" copies the newly-compiled binaries from the current directory to the appropriate system directories. This last step will usually copy the binaries (executables) to directories only writeable by root, as such you will need root permissions to perform this step.



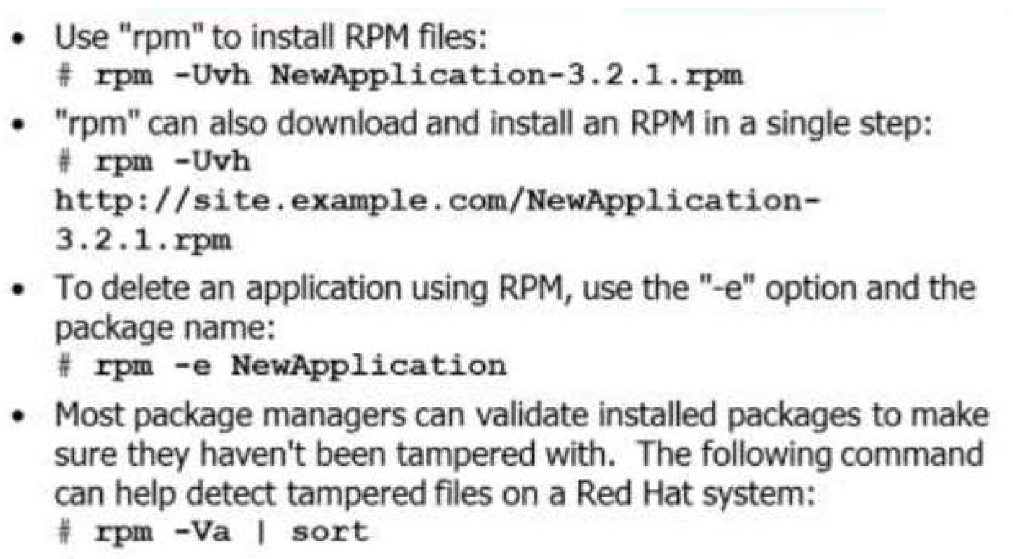
- These commands are often combined into a single command:
  ```
  # ./configure && make && make install
  ```
  - The double ampersand is a conditional operator that says "IF the first command succeeds, THEN execute the second command"
- To remove software installed from source, enter the original source directory and run:
  ```
  # make uninstall
  ```
  - This will not always work

These three commands are commonly joined together on a single line using double ampersands (&&) like this:

```
# ./configure && make && make install
```

The double ampersand is a conditional operator that says "IF the first command succeeds, THEN execute the next command".

To uninstall a program that was compiled from source, enter the original source directory and type "`make uninstall`". Unfortunately, not all software contains this feature in its Makefile.



- Use "rpm" to install RPM files:
  ```
  # rpm -Uvh NewApplication-3.2.1.rpm
  ```
- "rpm" can also download and install an RPM in a single step:
  ```
  # rpm -Uvh
  http://site.example.com/NewApplication-
  3.2.1.rpm
  ```
- To delete an application using RPM, use the "-e" option and the package name:
  ```
  # rpm -e NewApplication
  ```
- Most package managers can validate installed packages to make sure they haven't been tampered with. The following command can help detect tampered files on a Red Hat system:
  ```
  # rpm -Va | sort
  ```

RPM Examples

To install an RPM package on a Red Hat-based system, use the "rpm" command as follows:

```
# rpm -Uvh NewApplication-3.2.1.rpm
```

The "U" means "install or upgrade", the "v" means to print more verbose information, and the "h" means to print a progress bar during the install.

Red Hat systems can download and install an RPM package in one step using a command like this:

```
# rpm -Uvh http://site.example.com/NewApplication-3.2.1.rpm
```

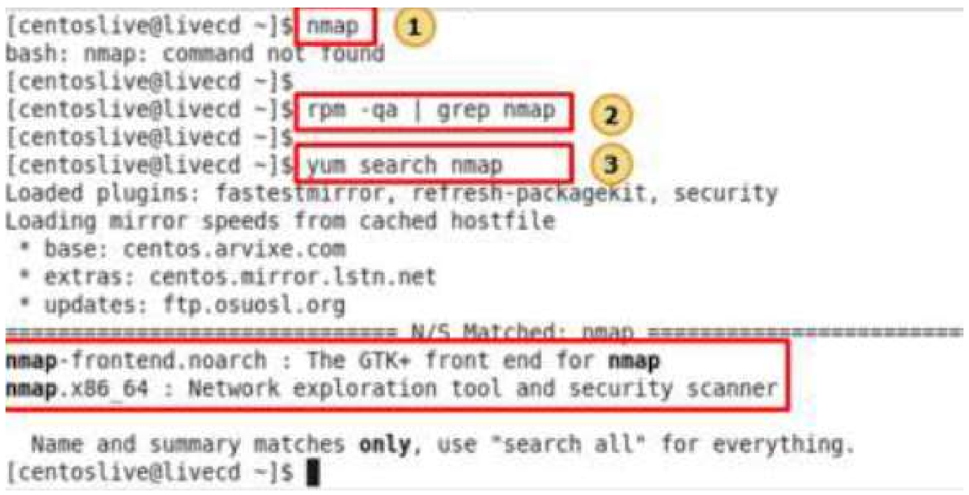
That same application could then be removed from the system with the following command:

```
# rpm -e NewApplication
```

If an attacker compromises a machine and modifies a file that belongs to an RPM package, then the following command can help detect that:

```
# rpm -Va | sort
```

The "V" means to verify packages, and the "a" means "all packages".



```
[centoslive@livecd ~]$ nmap
bash: nmap: command not found
[centoslive@livecd ~]$
[centoslive@livecd ~]$ rpm -qa | grep nmap
[centoslive@livecd ~]$
[centoslive@livecd ~]$ yum search nmap
Loaded plugins: fastestmirror, refresh-packagekit, security
Loading mirror speeds from cached hostfile
 * base: centos.arvixe.com
 * extras: centos.mirror.lstn.net
 * updates: ftp.osuosl.org
============================== N/S Matched: nmap ========================
nmap-frontend.noarch : The GTK+ front end for nmap
nmap.x86_64 : Network exploration tool and security scanner

  Name and summary matches only, use "search all" for everything.
[centoslive@livecd ~]$
```

Exercise: Package Repositories (1)

1. Try to start a network scanning tool called nmap:

   ```
   # nmap
   ```

   You should receive a "command not found" error.

2. Let's verify that it's not installed by querying the RPM database:

   ```
   # rpm -qa | grep nmap
   ```

   There should be no output, verifying that the nmap RPM has not been installed.

3. Use the "yum search" command to verify that the repositories have nmap:

   ```
   # yum search nmap
   ```

   Note: This requires a working internet connection

For our Linux use the apt-get install (Instead of yum that is for Centos)