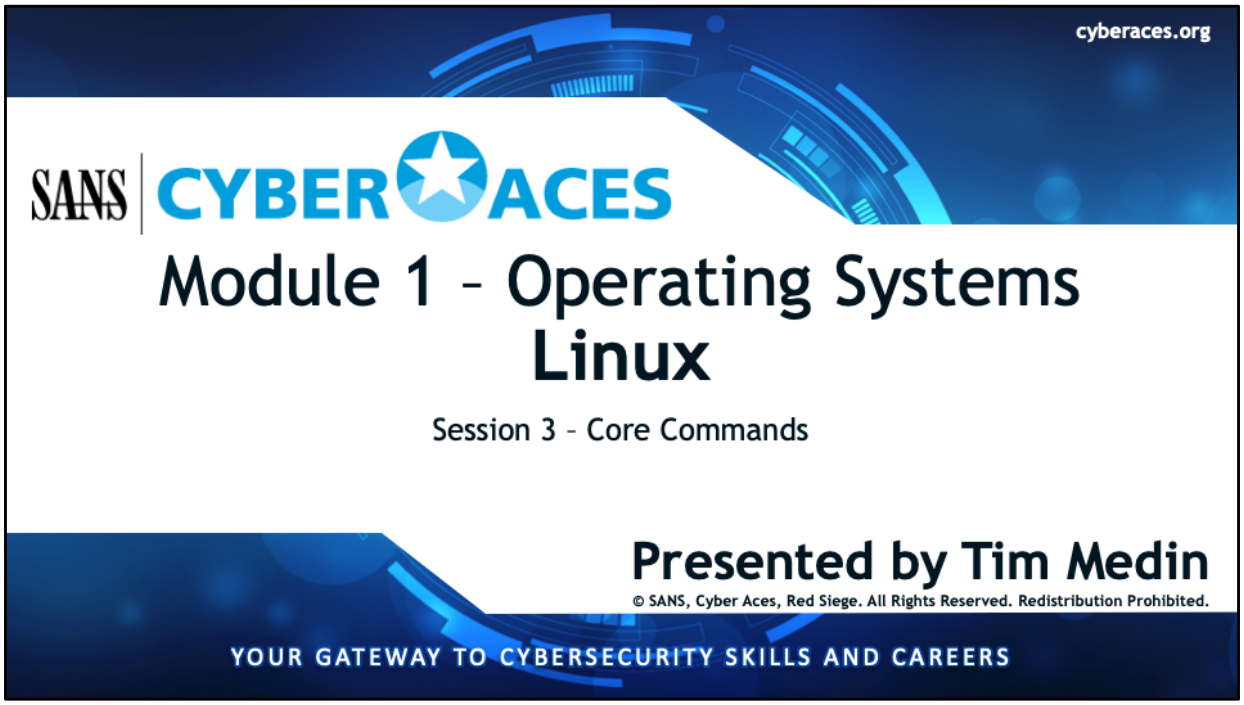
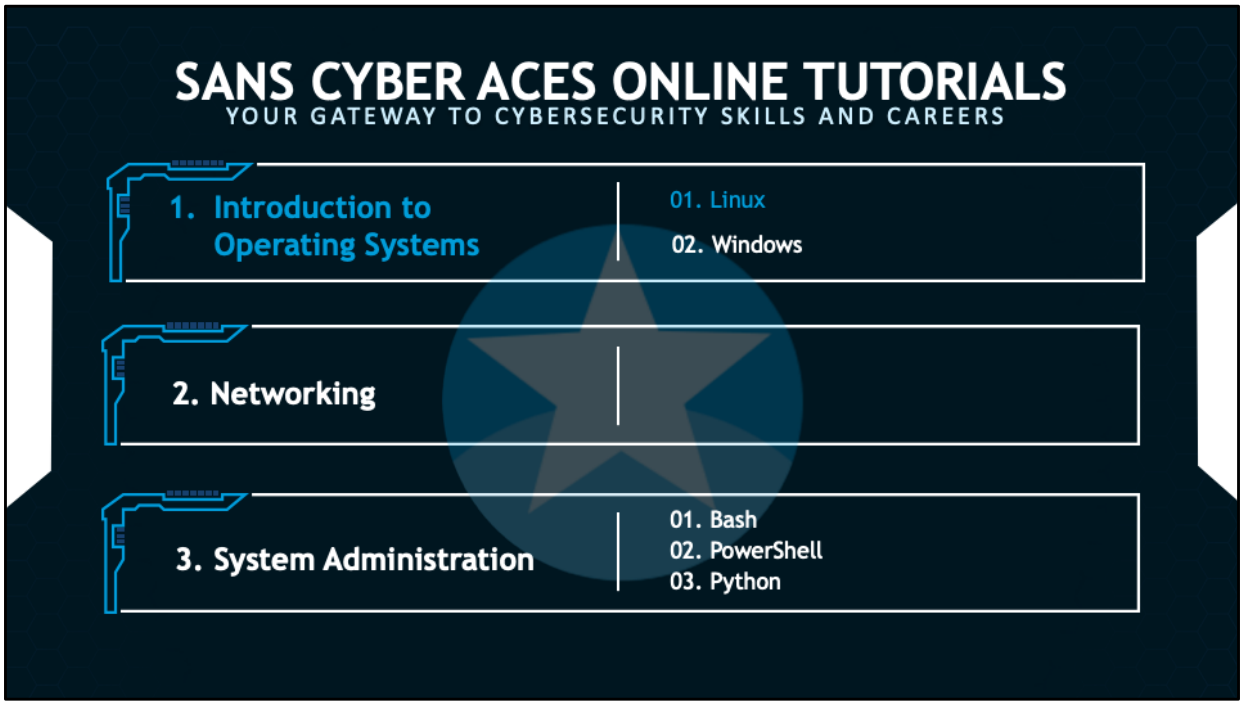


Welcome to Cyber Aces Online, Module 1 - Linux! This module provides a brief introduction to the Linux command on our CentOS VM.

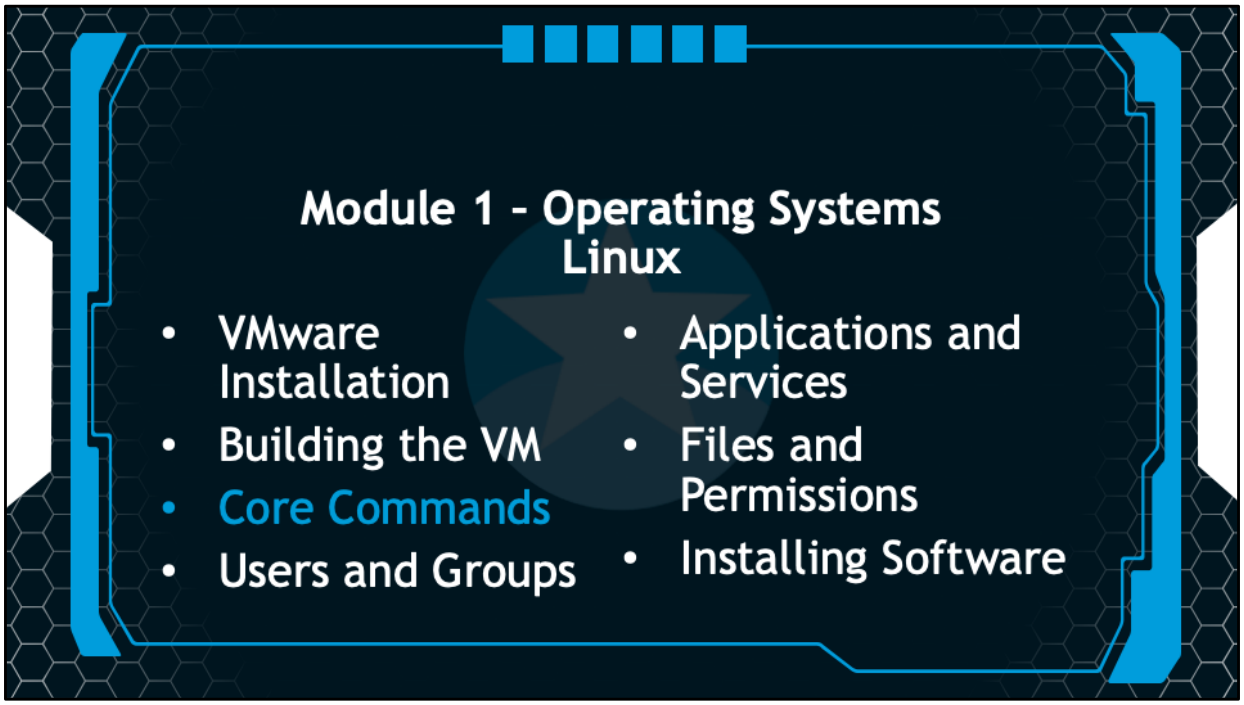
Content in this session has been developed by Tom Hessman, Tim Medin, Mark Baggett, Doug Burks, Michael Coppola, Russell Eubanks, Ed Skoudis, and Red Siege.



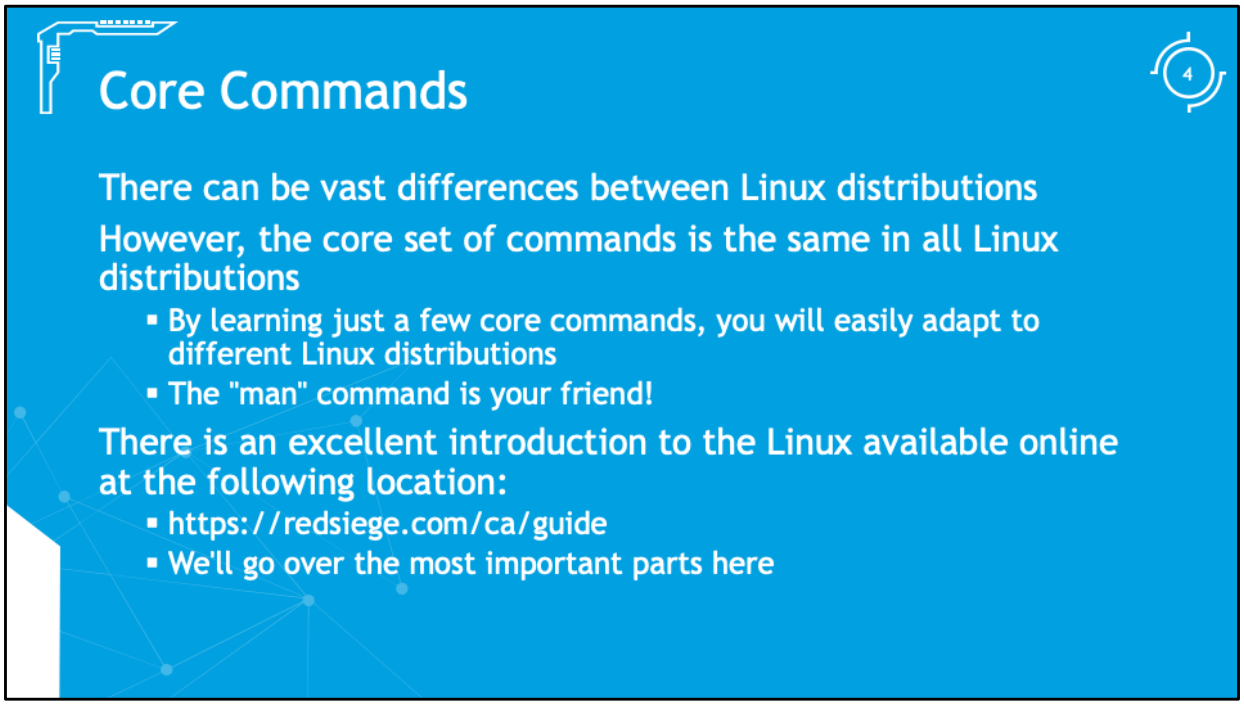
This training material was originally developed to help students, teachers, and mentors prepare for the Cyber Aces Online Competition. This module focuses on the basics of what an operating systems is as well as the two predominant OS's, Windows and Linux. This session is part of Module 1, Introduction to Operating Systems. This module is split into two sections, Linux and Windows. In this session, we will continue our examination of Linux.

The three modules of Cyber Aces Online are Operating Systems, Networking, and System Administration.

For more information about the Cyber Aces program, please visit the Cyber Aces website at https://CyberAces.org/.

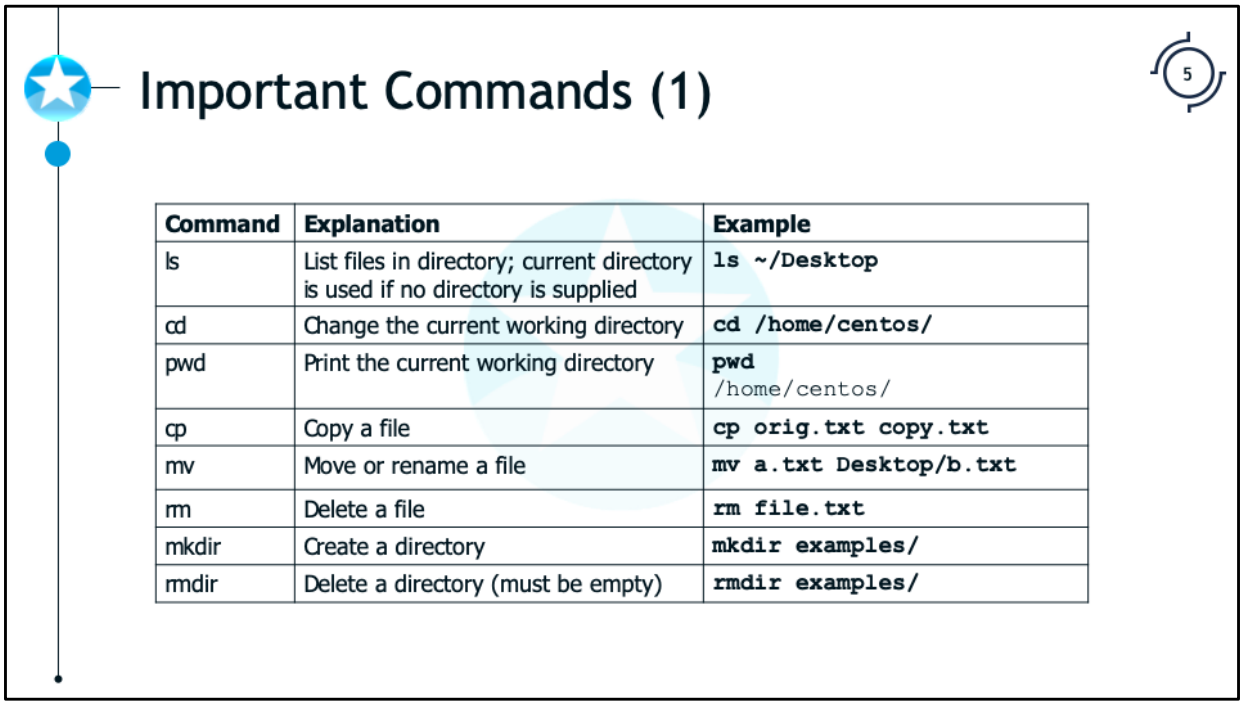


In this session we will cover the Linux's core commands.



The difference between Linux distributions can be significant. New users to Linux distributions often struggle with the fact that the command they just learned in one Linux variant may be completely different in the next. However, by memorizing just a few commands, such as "man -k", "apropos", "find", "locate", and "cat", along with the basic directory structure, you will be able to quickly adapt to the changes in these variants.

Many Linux commands perform operations similar to those of Windows commands, such as creating, viewing, and deleting directories and files, and some even share similar names. For instance, the command "cd" is used to change directory on both operating systems. The web site https://redsiege.com/ca/guide is a valuable resource, offering exemplified references of basic Linux commands to beginners. If you ever need help understanding a new command, simply type "man newcommandname"!



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 ..**).

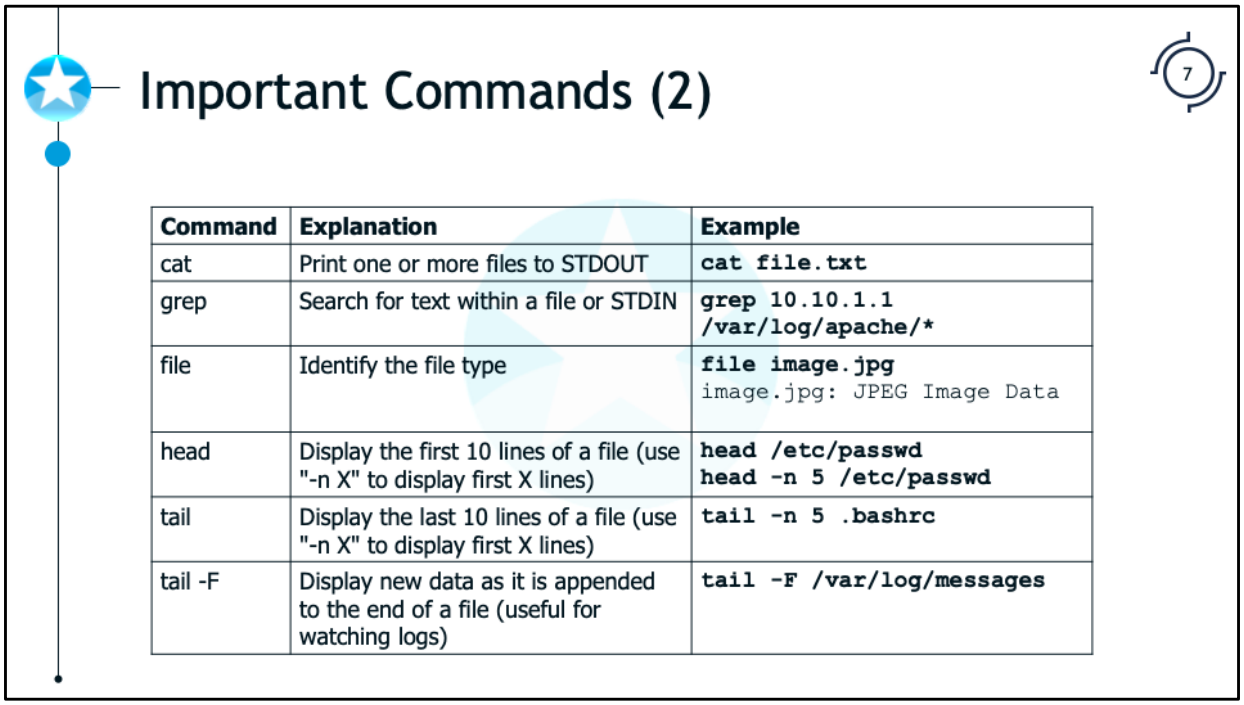
pwd (print working directory): Print the current working directory (the directory you are currently in).

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.



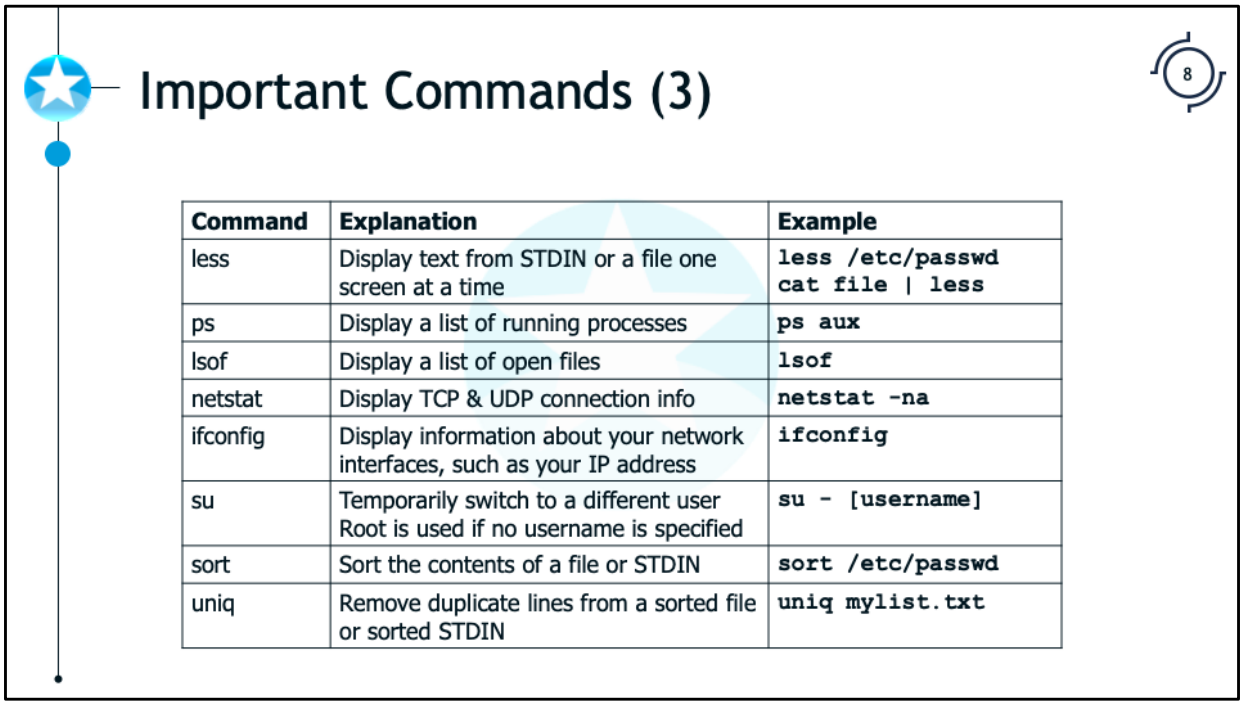
cat (concatenate): Print one or more files and print to STDOUT (normally the screen). It's called "concatenate" because it can be used to combine multiple files into one stream, such as: **`cat *.log > combined-log`**.

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.



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).

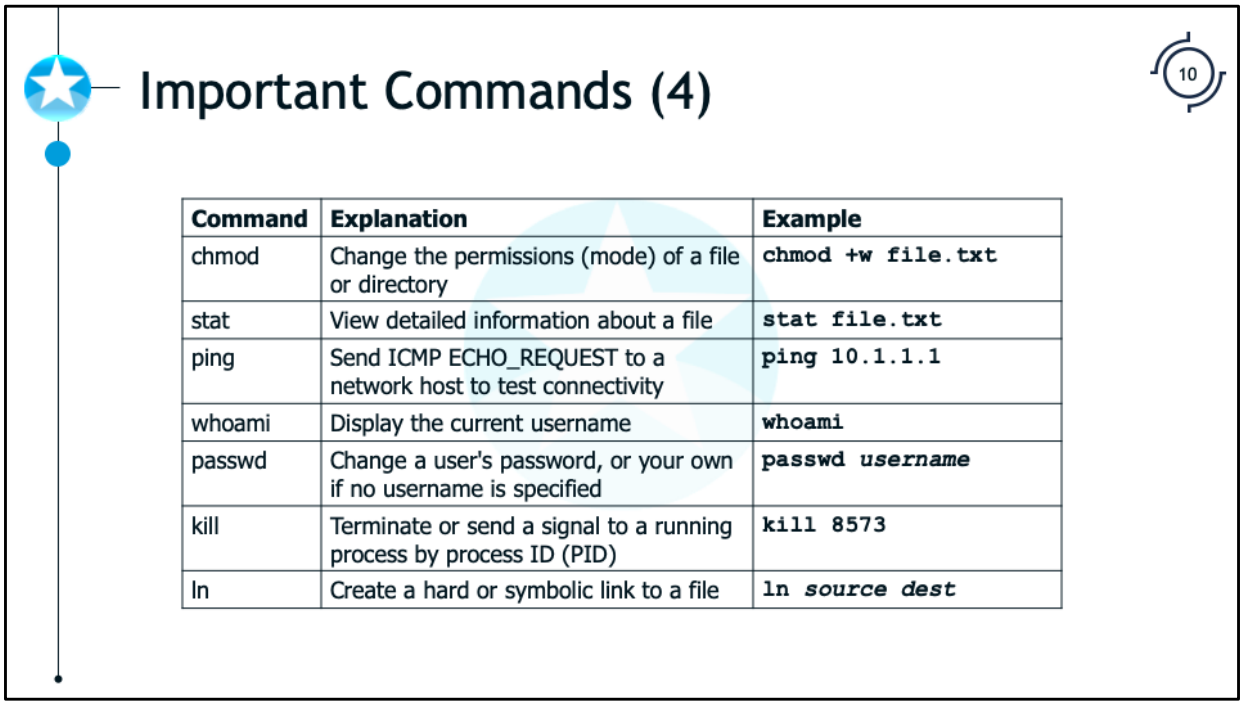
ps: Display information about running processes. The "aux" options indicate to list all running processes in the system, instead of only processes from the current shell. Output from "ps" is often piped into grep to search for instances of a particular program.

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.



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.

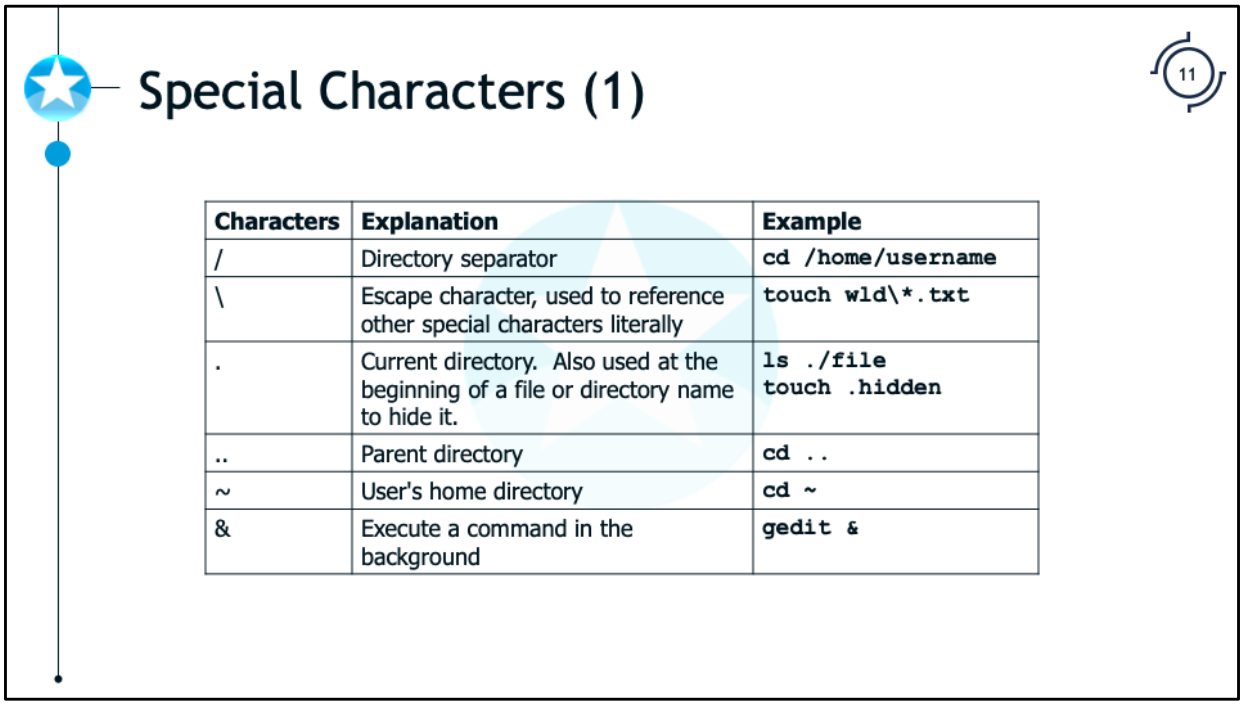
ping: Send ICMP ECHO_REQUEST packets to a network host and wait for responses to test network connectivity. The "ping6" program can be used for IPv6 addresses.

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.



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.

Special Characters (2)

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".

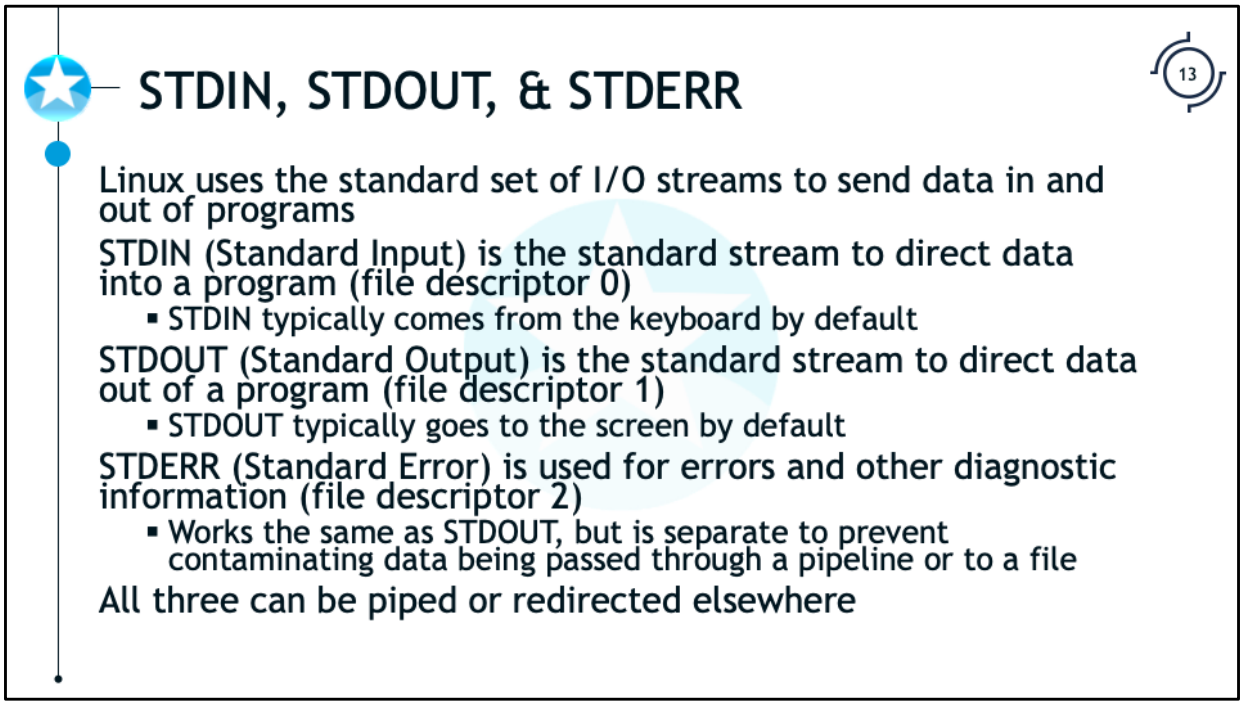
? (question mark): A wildcard used to represent a single character in a filename. For example, running **`ls pic?.jpg`** would match "pic1.jpg" and "pic2.jpg", but not "pic23.jpg" or "pic.jpg".

[] (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.

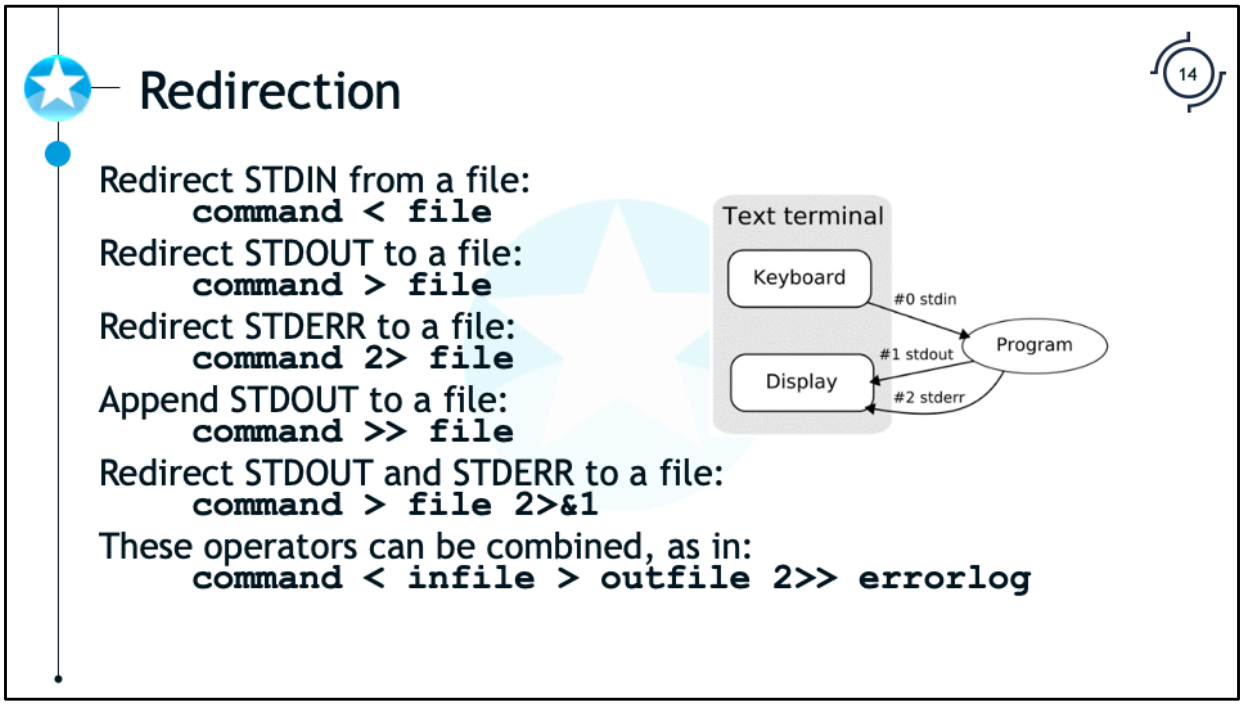
; (semicolon): A command separator that can be used to run multiple commands on a single line, unconditionally.

&& (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 (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.



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
```

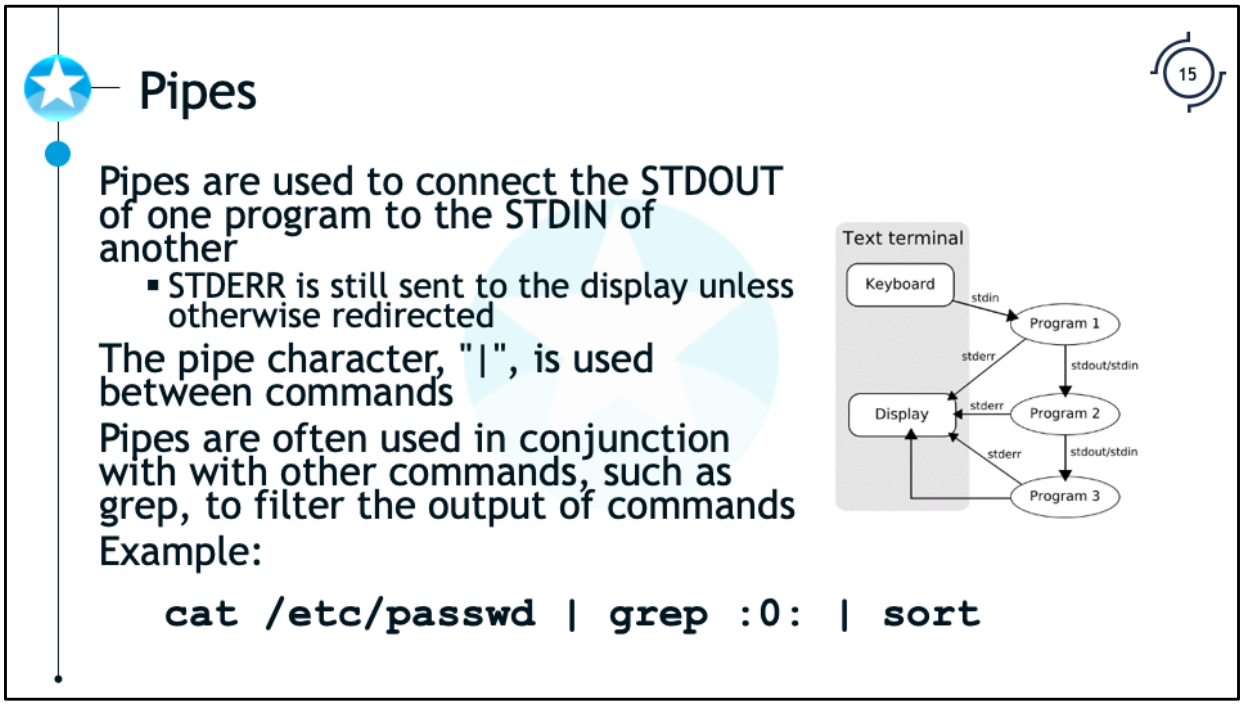
Redirect STDOUT and STDERR to a file (the "2>&1" sends 2 to file descriptor 1, which is STDOUT):

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

These operators can be combined, as in:

```
$ command < infile > outfile 2>> errorlog
```

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, 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.

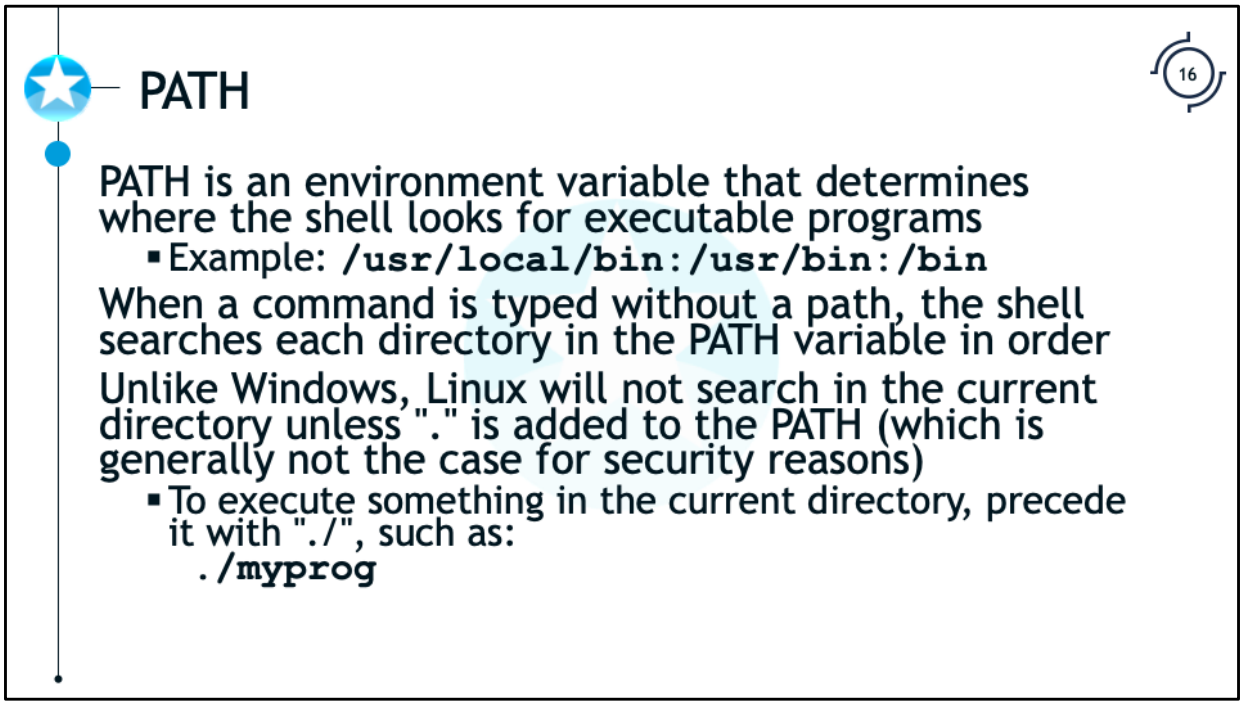
For example, the following command will read in the list of users on the system, search them for the string ":0:" (identifying users with UID or GID 0), and then sort them alphabetically:

```
$ 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).

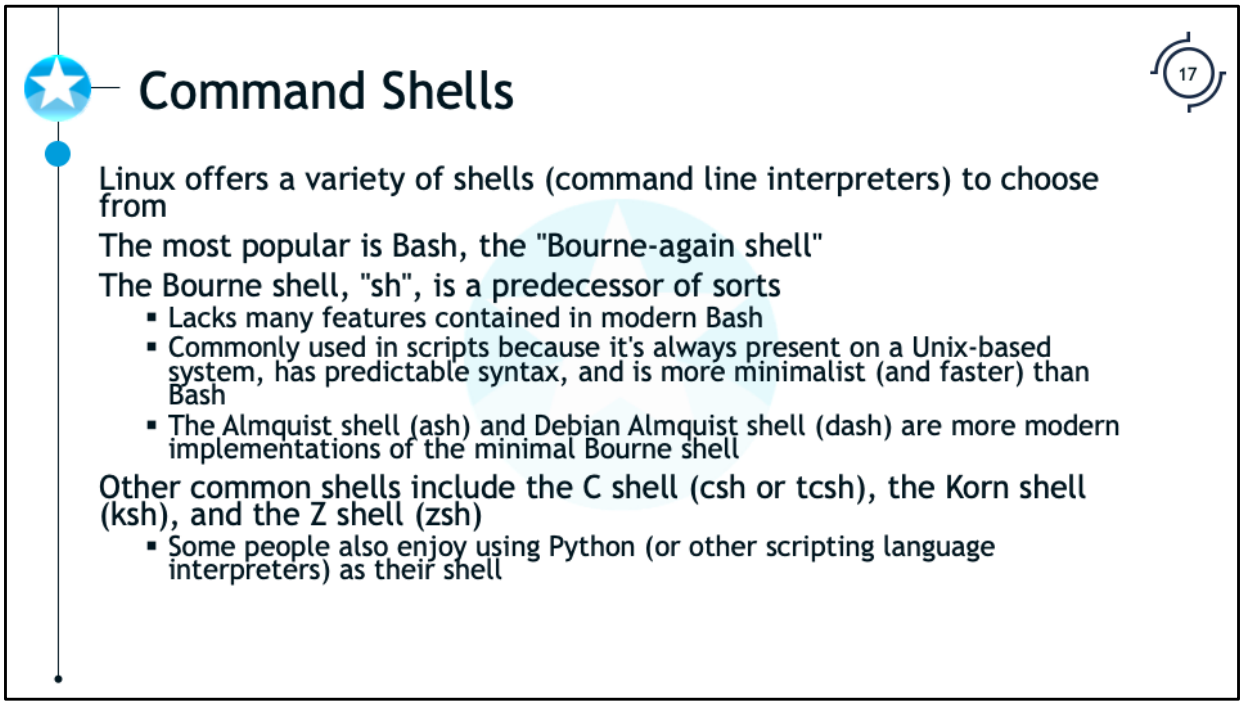


"PATH" is an environment variable that determines where the shell looks for executable programs. When a command is typed without a path to the executable, the shell searches each directory in the PATH variable in order. For example, if the PATH variable is set to "/usr/local/bin:/usr/bin:/bin" and you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

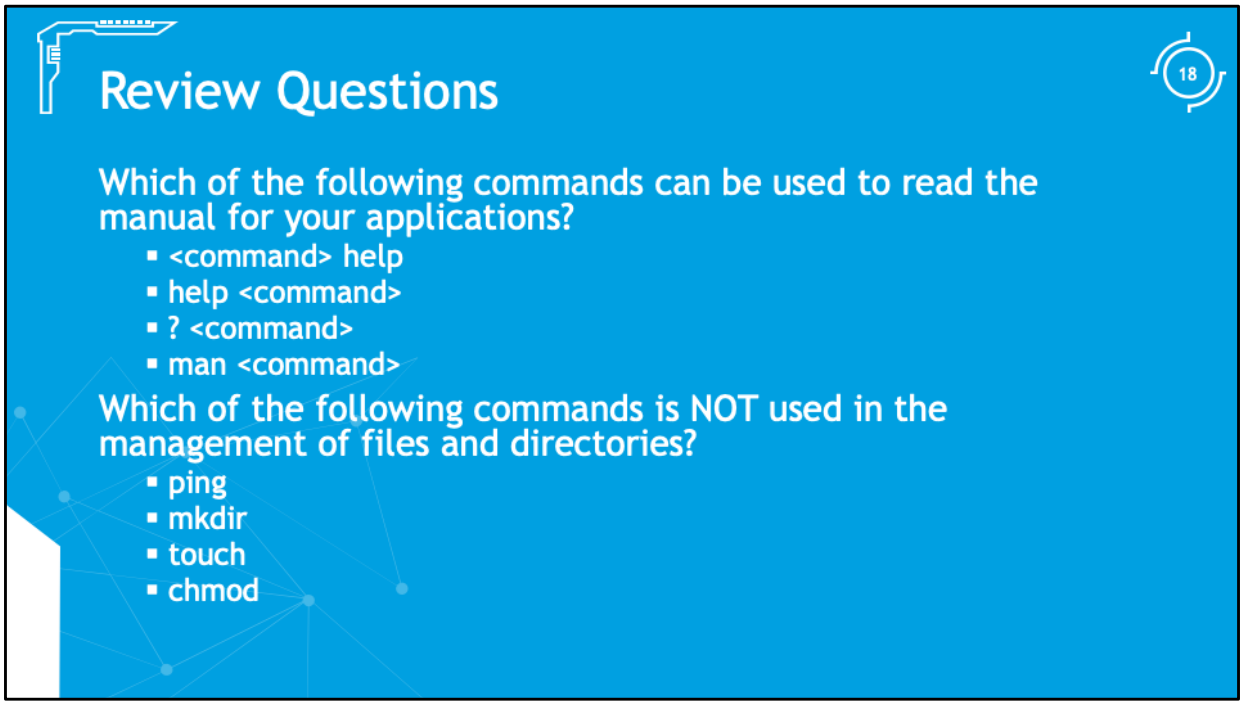
If you do need to run an executable that is in the current directory, but not in the path, precede it with "./".

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



Linux offers a 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.

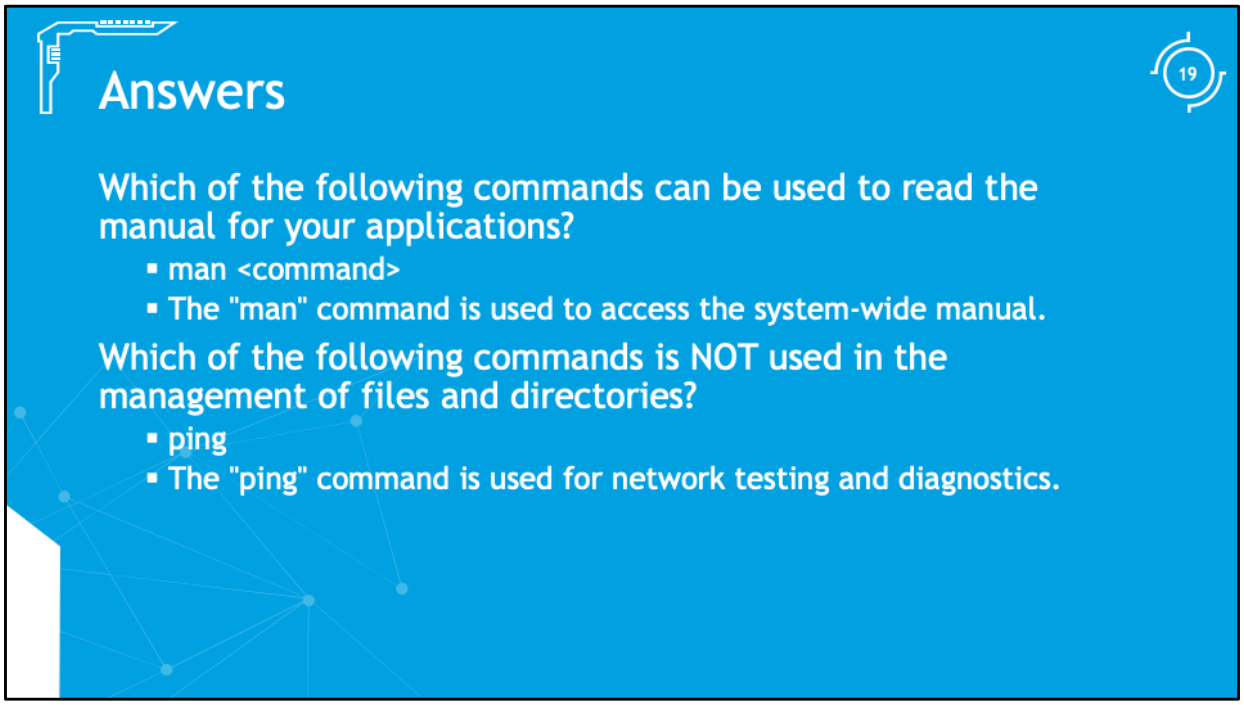


Which of the following commands can be used to read the manual for your applications?

<command> help
help <command>
? <command>
man <command>

Which of the following commands is NOT used in the management of files and directories?

ping
mkdir
touch
chmod



Which of the following commands can be used to read the manual for your applications?

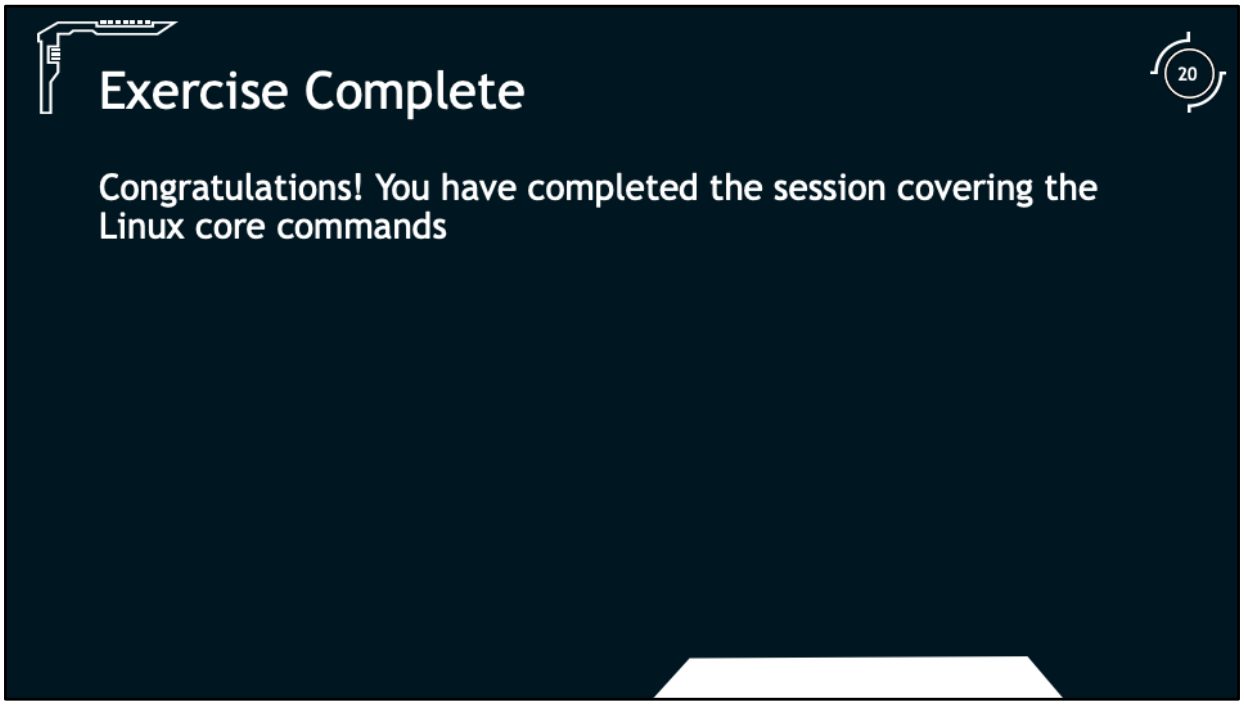
man <command>

The "man" command is used to access the system-wide manual.

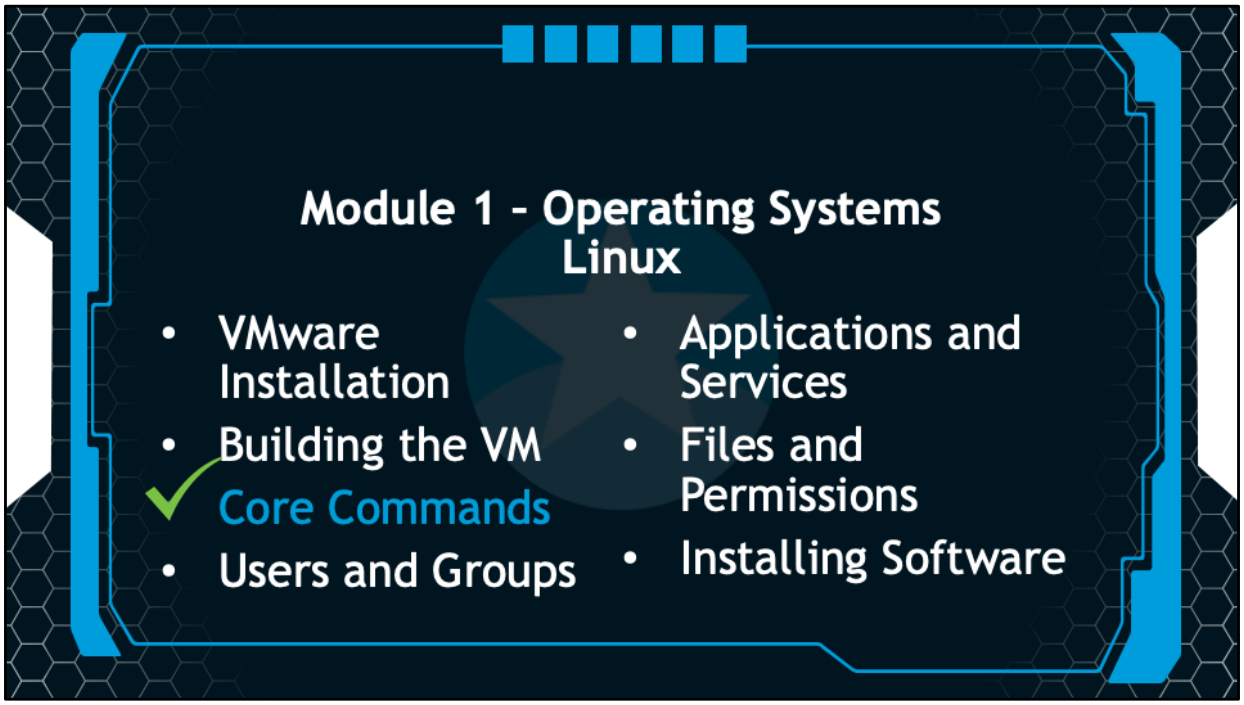
Which of the following commands is NOT used in the management of files and directories?

ping

The "ping" command is used for network testing and diagnostics.



Congratulations! You have completed the session covering the Linux core commands.



In the next session we will discuss Linux users and groups.