launch_code

John Woolbright, Technical Curriculum Developer

June, 15, 2022

Bash: Streams, Redirection, and Pipes

Types of Streams

STDIN: Standard Input - 0 (file descriptor)

STDOUT: Standard Output - 1 (file descriptor)

STDERR: Standard Error - 2 (file descriptor)

Standard Input

All Bash commands are a form of STDIN

Standard Output

STDOUT is always displayed inside of the terminal window by default

Should you ever wish for STDOUT not to be displayed this can be accomplished in different ways

Sometimes a command allows -s as a "silent" option

You can also redirect the stream into an alternate location

Standard Error

Similar to STDOUT, STDERR is also displayed inside of the terminal window by default

STDERR is commonly the result of an incorrect bash command, incorrect arguments, or incorrect options provided

Redirection

- Redirect Write >
- Redirect Append >>

Redirect Write with STDOUT

The > operator is used to redirect STDOUT and write to a new location

Using the redirect write option will overwrite the contents of the target location

Redirecting the STDOUT will also prevent the STDOUT of the command to be displayed inside of the terminal window

Redirect Append with STDOUT

The >> operator is used to redirect STDOUT and append to a provided location

Using the redirect append option will append the contents of the target location

This means that the STDOUT will be added to a given file in addition to what is already inside

As mentioned in the previous slide this will also prevent the STDOUT from being displayed inside of the terminal window

Redirect STDERR

STDERR can also be redirected using its file descriptor

To write STDERR to a new location you would use 2>

To append STDERR to a new location you would use 2>>

Pipes

The Bash pipe operator

These *|* are *|* Pipes *|*

Allows the user to take STDOUT from the first command and use it as STDIN for the next

Piping allows the user to chain multiple commands together for more complex commands

Bash File System

The Linux File System

The file system is a part of the Linux kernel.

In Linux everything is a file. That is to say directories, files, links, and a few other Linux tools are all files and are a part of the file system.

Knowing the basics of the Linux FS hierarchy, Bash shell FS navigation commands, and how to perform actions on files is a fundamental aspect of using Linux.

File System Hierarchy

The Linux FS begins with the root (`/`) directory. This is the top level directory that contains all other directories and files that are a part of the operating system.

If you understand the general purpose of top level directories inside of the root directory, you will have a good idea where various files live.

Required root Directories

According to the Linux Documentation Project: https://tldp.org/ the Linux FS is required to have the following directories inside of the root directory.

- /bin, /boot, /dev
- /etc, /lib, /media
- /mnt, /opt, /sbin
- /srv, /tmp, /usr

In this class you are expected to know the purpose and use of the files in /bin, /etc, /usr and the often utilized /home directories.

/bin Essential Command Binaries

Contains useful commands that are used by both the system administrator and non-privileged users.

Contains shells like bash. Contains commonly used shell commands like cp, mv, rm, cat, ls.

The tools found in /bin are separate from many of the other user tools as they are crucial for servicing the operating system if other areas become corrupted.

/etc Host-specific system config

The container for all system related configuration files. These files are used to control the operation of all programs.

Example files hostname (name of computer), timezone (timezone of computer), environment (the contents that control the $PATH shell variable).

/usr User binaries, docs, libraries, header files, etc

The container for programs and files that are shared across all users of the computer.

Many of the applications we will learn about in this class are found in /usr.

Examples include nano, python3, man, and which.

/home multi-user data and apps

Linux distributions are allowed to add additional directories to the root directory. Ubuntu, and some other Linux distros add /home.

/home is the location of all individual user data and applications.

Some Linux distributions create the /home directory within the /usr directory.

Other root level Directories

Check the slides below for the top level description of the remaining TLDP top level directories.

We will not be covering the details of these directories, however you can learn more at the Linux Filesystem Hierarchy from TLDP: https://tldp.org/LDP/Linux-Filesystem-Hierarchy/Linux-Filesystem-Hierarchy.pdf.

Examples

/boot: Static files of the bootloader

/dev: Device files

/lib: Essential shared libraries and kernel modules

/media: Mount point for removable media

/mnt: Mount point for mounting a file system temporarily. (Flash drive, external hard drive, etc..)

Examples

/opt: Add-on application software packages

/sbin: Essential system binaries

/srv: Data for services provided by the system

/tmp: Temporary files

Bash File System Command Review

pwd: print working (current) directory

Ls: list contents of current directory

These two commands will help you get your bearings as you move away from the home directory

Navigating the File System

Changing the working (current) directory is a common and useful action while in a Bash shell.

You can change the current directory with the cd shell builtin command.

It takes an optional argument in the form of the directory which is used to update the current working path.

Change Directory Examples

cd /home/student/Documents: absolute path

cd Documents: relative path

cd .Documents: relative path

cd ~: the ~ key is a shortcut for $HOME

cd : the default argument is $HOME

cd ..: change to the parent directory

Creating Directories

mkdir [new-dir-name]: make directory

mkdir works with both relative and absolute paths

Creating Empty Files

touch new-file.txt: create a new empty file named “new-file.txt” in the current working directory

touch can be used to create new empty files, but it's main purpose is to update an existing file's timestamp.

You can try this out by creating a new file with touch example-file checking the last file modified date with ls -l and then waiting a few minutes and then running touch example-file again.

Displaying File Contents

Reading the contents of a file is always handy. In a terminal you can either take the entire contents and dump it into STDOUT with cat.

Or you can break the output into chunks and scroll through them manually with less.

Searching for Files

Find [location] -name [file-name]

Moving Files and Directories

mv file new-location/: move file to new-location/file

Using mv to rename Files

You can use the mv command to rename files and directories in place, or change their name when moving to a new location.

mv file file2: will rename file to file2

mv file Documents/file2: will move and rename file to Documents/file2

Deleting Files

rm file-name: relative or absolute path

rm will also delete directories, but it must first delete any contents inside of the directory.

You can trigger this behavior with the -r option.

rm -r Documents: delete the Documents directory and all files/directories in the Documents directory.

You can stop it from asking about every document by adding the --force option. This can cause some nasty effects if used on the wrong directory.

Editing contents of a File

nano is a terminal text editing program

nano file-name will open the existing file-name or create a new file and open it for editing/adding content

~/.bashrc

File run for every new shell session initiated by user.

This is where you can add things to permanently add them to your shell.

You can also add any shell customizations here.

Bash: File Permissions

Linux File Permissions

All files have permissions for different types of classifications

The classifications are Owners, Groups, and Others

Available Permissions

Read, Write, Execute, None

Read: User, Group, and Other can Read the file

Write: User, Group, and Other can Write to the file

Execute: User, Group, and Other can execute the file

None: None of the above permissions

Possible Permission Combinations

Read only, Write Only, Execute only

Read and Write

Read and Execute

Write and Execute

Read, Write, and Execute

None

View Permissions

You can use the -l option with the ls command for a long listing format of files within a given directory

You can expand even further and provide the -a option in addition to -l for all files (including hidden)

Changing File Permissions

The chmod command allows you to change file permissions for a file or directory

chmod [OPTION] [file-name]

Read is represented by the numeric value 4

Write is represented by the numeric value 2

Execute is represented by the numeric value 1

None is represented by the numberic value 0

chmod Command Example

chmod 444 example-file-name

The above command would provide the Owner, Group, and all Other users with access to the "example-file-name" file with Read only permissions

Providing Multiple Permissions

You are able to use the numeric values in addition to one another to provide multiple permissions for any given Owner, Group, and Other users

Read + Execute = 5

Read + Write = 6

Read + Write + Execute = 7

Changing File Ownership

The **chown** command allows you to change file ownership for a file or directory

chown [OPTION] [OWNER][:[GROUP]] [file-name]

Changing User Ownership

chown new-user example-file

The above command would change the ownership of the example-file from its current owner to "new-user"

Changing Group Ownership

chown :new-group example-file

The above command would change the ownership of the example-file from its current group to "new-group"

Changing User and Group Ownership

chown new-user:new-group example-file

The above command would change the ownership of the example-file from its current owner to "new-user" and group to "new-group"

launch_code

launchcode.org