ITE315 Module 1 Part B - Bash

Athens State University

Contents

1 The Command Processor

The Anatomy of The Command Line Interface

- **Command Processor**: Computer program designed to interpret a sequence of lines of text entered from a data stream

  Although most users think of the shell as an interactive command interpreter, it is really a programming language in which each statement runs a command. Because it must satisfy both the interactive and programming aspects of command execution, it is a strange language, shaped as much by history as by design. - Kernighan & Pike, *The UNIX Programming Environment*

The Anatomy of The Command Line Interface

- **Prompt**: Generated by the CLI to provide context to the user
- **Command**: Entered by the user to cause something to happen
- **Parameters**: Optional components that adjust command behavior

Commands are usually in one of three classes:

- Internal: recognized and processed by the CLI and not dependent upon any external executable file
- Included: A separate executable generally considered part of the operating environment and always included with the OS
- External: External executable files that are not part of the basic OS but are added by other parties for specific purposes and applications

Operating Systems and Command-Line Interface

Windows	Windows Command Prompt, Powershell, and (indirectly) Bash
macOS	Bash, tcsh, zsh
Linux	Bash, tcsh, zsh

Command-line interpreters have evolved through two lineages over the history of personal computing. On one path we have shells that evolved from the command line interface introduced with the UNIX operating system in the early 1970s. Modern UNIX-based operating systems such as Linux, macOS, and FreeBSD use the Bourne Again Shell, which is an evolution of the Bourne shell from 1980s era UNIX System V OS, which was an enchanced version of the original UNIX shell CLI.

On the other hand, we have the command-line interface in Windows. This program was adopted from the CP/M operating system, which was inspired by the command-line interpreter on the DEC PDP-8 and PDP-11 mini-computers. This evolved into the `cmd.exe` utility that we now use on Windows 7 and Windows 10. As this CLI is rather primitive and doesn't support automation very well, Microsoft is now deploying in beta test a replacement that combines the best features of the `cmd.exe` and `Powershell` interfaces.

In this course, we will focus on two command-line interpreters: Bash and Powershell.

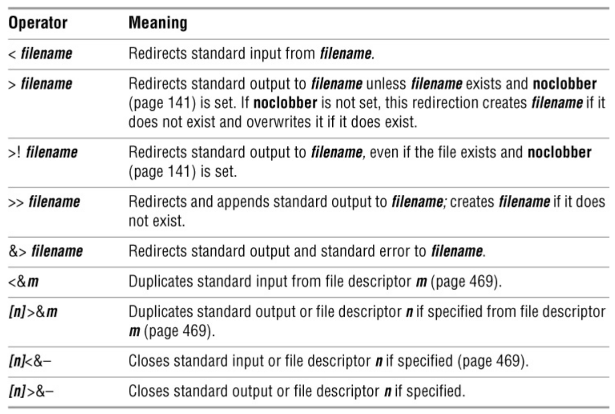
2 Working With Bash

When you use a shell as a command interpreter, you can customize the environment you work in. You can make the prompt display the name of the working directory, create a function or an alias for cp that keeps it from overwriting certain kinds of files, take advantage of keyword variables to change aspects of how the shell works, and so on. You can also write shell scripts that do your bidding—anything from a one-line script that stores a long, complex command to a longer script that runs a set of reports, prints them, and mails you a reminder when the job is done. More complex shell scripts are themselves programs; they do not just run other programs.

The Login Shell

- **Login shell**: The first shell that display when you log in to a system from system console, virtual console, remotely using `ssh`, or via a Terminal application.

Input and Output



Operator	Meaning
< ***filename***	Redirects standard input from ***filename***.
> ***filename***	Redirects standard output to ***filename*** unless ***filename*** exists and **noclobber** (page 141) is set. If **noclobber** is not set, this redirection creates ***filename*** if it does not exist and overwrites it if it does exist.
>! ***filename***	Redirects standard output to ***filename***, even if the file exists and **noclobber** (page 141) is set.
>> ***filename***	Redirects and appends standard output to ***filename***; creates ***filename*** if it does not exist.
&> ***filename***	Redirects standard output and standard error to ***filename***.
<&***m***	Duplicates standard input from file descriptor ***m*** (page 469).
[n]>&***m***	Duplicates standard output or file descriptor ***n*** if specified from file descriptor ***m*** (page 469).
[n]<&–	Closes standard input or file descriptor ***n*** if specified (page 469).
[n]>&–	Closes standard output or file descriptor ***n*** if specified.

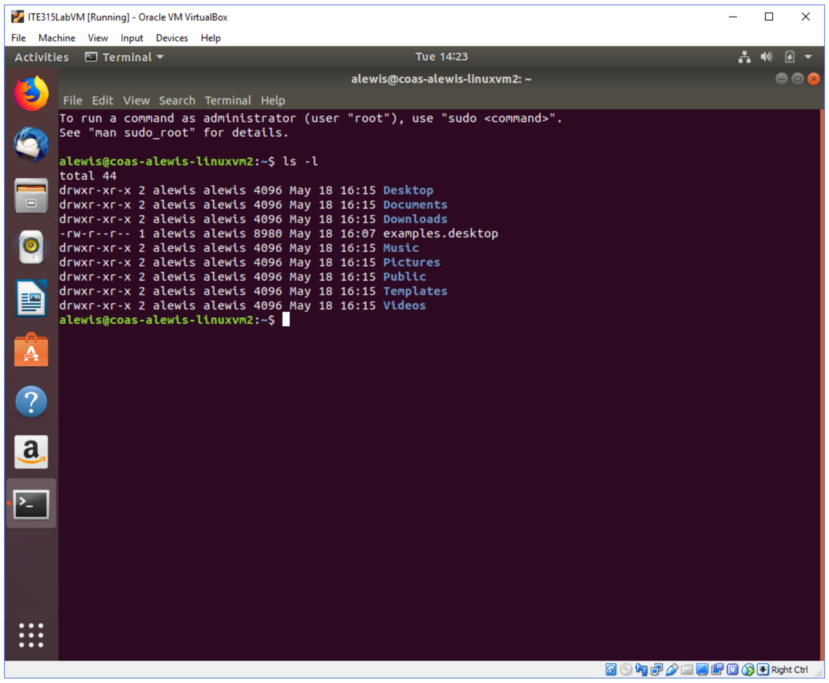
3 Labs 2, 3, and 4: Command Lines and FIle Management

Labs 2, 3, and 4: Command Lines and File Management

- At this point, we will digress into three labs dealing with navigating the file system using the Bash shell

4 Access Permissions

Files and Access Permissions


ITE315LabVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Terminal
Tue 14:23
alewis@coas-alewis-linuxvm2: ~
File Edit View Search Terminal Help
Right Ctrl

```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

alewis@coas-alewis-linuxvm2:~$ ls -l
total 44
drwxr-xr-x 2 alewis alewis 4096 May 18 16:15 Desktop
drwxr-xr-x 2 alewis alewis 4096 May 18 16:15 Documents
drwxr-xr-x 2 alewis alewis 4096 May 18 16:15 Downloads
-rw-r--r-- 1 alewis alewis 8980 May 18 16:07 examples.desktop
drwxr-xr-x 2 alewis alewis 4096 May 18 16:15 Music
drwxr-xr-x 2 alewis alewis 4096 May 18 16:15 Pictures
drwxr-xr-x 2 alewis alewis 4096 May 18 16:15 Public
drwxr-xr-x 2 alewis alewis 4096 May 18 16:15 Templates
drwxr-xr-x 2 alewis alewis 4096 May 18 16:15 Videos
alewis@coas-alewis-linuxvm2:~$
```

Consider again the long output from the `ls` command. One part of that listing is the access permissions on files. This tells you who can do what things to a file.

Access Permissions

```
drwxrwxrwx
```

- Permissions are organized into three groups:
  - **Owner**: Applies only to the owner of the file or directory
  - **Group**: Applies only to the group to which the file or directory has been assigned
  - **World**: Applies to all other users on the system

Access Permissions

```
drwxrwxrwx
```

- Each permission group is composed of three bits:
  - **Read**: Can a user read the contents of a file
  - **Write**: Can a user write the contents of a file
  - **Execute**: Can a user execute this file or view the contents of a directory

Advanced Permissions

- **d**: directory
- **l**: this is a link; i.e., an alias
- **s**: Special permission for administrative use
- **t**: The “Sticky Bit”

The setuid/setguid permissions are used to tell the system to run an executable as the owner with the owner’s permissions.

Be careful using setuid/setgid bits in permissions. If you incorrectly assign permissions to a file owned by root with the setuid/setgid bit set, then you can open your system to intrusion.

You can only assign the setuid/setgid bit by explicitly defining permissions. The character for the setuid/setguid bit is s.

The sticky bit can be very useful in shared environment because when it has been assigned to the permissions on a directory it sets it so only file owner can rename or delete the said file

Changing Permissions

- The Linux `chmod` allows you to adjust the permissions of a file
- You can do this using symbolic representation of the permissions
- Or you can specify the permissions in binary

5 Manipulating The Directory Stack

The Directory Stack

- The shell keeps a *stack* of the directories that you have visited during a login session
  - Think about how plates are stacked at your favorite buffet restaurant.
  - Last In, First Out
- This allows you to easily move up and down the directory tree

Useful Dir Stack Commands

- `dirs`: display the contents of the directory stack
- Recall that the tilde represents your home directory
- `pushd`: Push a directory onto the stack
- `popd`: Pop a directory off of the stack
- `cd` : Change the current directory to be the previous dir

6 Process and Job Control

Processes

- Commands are executed in an operating system *process*
- **Job**: A process running a pipeline
  - **Foreground job**: Each session has one foreground job, which is displayed on the screen
  - **Background job**: A session can have one or more *background jobs*, which run simultaneously to the foreground job and each other
- To run a job in the backround, place an ampersand (&) at the end of the command-line

Moving Jobs From The Foreground To The Background

- Typing the `CTRL-Z` key will *suspend* the current foreground job
- Then the `bg` command will restart a suspended background job
- The `fg` command will move a job from the background to be the currently running foreground job

Aborting A Background Job

- One can interrupt the foreground process by typing the `CTRL-C` command
- This issues a `kill` to the current foreground process
- You can explicitly kill a process using the `kill` command

Explicitly Listing Jobs and and Processes

- `jobs`: Displays a list of suspended and background jobs
- `top`: Interactively display processes
- `ps`: Display process status

7 Key Points

Key Points

- What is a command-line interpreter?
- The basic use of the Bash shell
- Efficiently using the command line