The Linux Command Line

Linux System Administration
COMP2018 Fall 2019

Command Line Concepts

- Programs which accepts commands in text format from an interactive user are called command line interface (CLI) programs
- You type a single line of text and when you press enter, the program interprets what you typed as a command and possibly some options to control it as well as possibly some data to feed to it
- That command is run (executed) and then you are asked for another command
- CLI programs normally present the user with a prompt in a fixed format to make it easier for a user to know when they can enter a new command
- Before you can start giving commands to a CLI program, you must first identify yourself to the system so that the CLI program can be granted appropriate permissions to access resources in the system

Login

- Logging into a Linux system is how an interactive user identifies and authenticates themselves before using the computer

- Some program listens for login attempts on whatever devices or connection types your system is configured to support - they collect your login name and pass that to the login program which collects your password and checks if you gave it correctly

- Your login name (user name or account name) identifies you to the system, and your password authenticates you (gives some proof you are who you claim to be) - this forms the basis for access control

- Your user account defines what program (or shell) starts after you successfully authenticate - this is the shell within which you can get your work done

Login Shell

- Your shell program defines how you interact with the computer (your interface)
- Shell programs have an environment to run in which defines what you can do and the interface defines how you do it
- When you quit your login shell, you are logging out
- Linux is a multiuser system, many users can login simultaneously, each user can login multiple simultaneous sessions

Remote vs. Local Login

- Login can be done using directly connected terminal hardware (a device with at least a keyboard and screen, or the special terminal called the console device)

- Login can be done over the network, using a protocol and toolset such as telnet, rlogin, or ssh

- When login happens over the network, the terminal device is virtual and called a pseudo-terminal

Remote Access - telnet

- telnet is a very simple and old program
- It connects to a network port, then prints on your screen more or less whatever the other ends sends and sends whatever you type to the other end
- If you run a telnetd process at the remote end, it will listen for these connections and when it gets one, it starts the login program with your network connection as the input and output for that login session
- There is no target authentication, no encryption, just direct connections with bytes going back and forth
- It is deprecated for login use, but is still valuable because of its simplicity for testing text-based protocols over the network
- The corresponding program which was created for copying files between machines was called ftp but used a separate protocol and connection

Remote Access rlogin/rsh/rcp

- rlogin replaced telnet for remote command line access by adding some authentication features to the network protocol
- This made it possible to do remote command execution without the user going through a login procedure, by using the rsh command
- It also made possible remote file copying without the user going through a login procedure using the rcp command
- These authentication features were easy to spoof and there is still no encryption for the connection
- rlogin/rsh/rcp are deprecated as insecure, and not useful for other tasks

Remote Access - ssh

- ssh is the current standard method of performing text-based authenticated and encrypted network connections to Linux systems
- It includes trivial server identification to alert the user to server spoofing
- It includes username based identification
- It includes password and/or crypto key based authentication
- It includes encryption of the communications passing over the network
- The ssh client program connects to an sshd server process to initiate a login session

Terminals

- The keyboard, mouse, and display screen of a computer is a physical terminal and has the special device name console
- Additional physical terminal devices may be connected using serial ports, they are known as tty devices (ref: the original teletype terminals)
- A network login connection has no physical device on the server, so the server creates a virtual terminal device for each connection, called a pty device (there are actually master and slave devices, called pty and pts respectively)
- Graphical terminal programs only create a visual interface for the user and use a typical virtual terminal on the local host
- Virtual machine software fakes the console device so that programs which expect a physical console device don't have problems

Command Line Users

- Users who login expect to get a program that lets them do whatever work they logged on to accomplish

- Some users are doing some specific task and can use a single-purpose program such as a Point-Of-Sale terminal program or data collection program for their shell

- Others are general purpose Linux users and want to be able to do whatever they want when they login

- They get a Linux shell program such as bash which allows them to run any other command on the system that they have privileges for - we call these command line users

- The 2 types of users that typically want to use a shell are administrators and developers

Shell Syntax

- For the remainder of this course, when we discuss command line users, we mean the command line interface offered by the bash shell program

- The general format of a command for bash is:

command [options] arguments

- bash commands, options, and arguments are all case-sensitive

- Most commands have a default mode of operation and can run without any options

- Some commands require arguments (or data), some do not, and some do not recognize any arguments

- Most commands support a help option e.g. -h or --help

File Storage

- The most common thing for any user to do is create, manipulate, and remove data stored in files
- Files are stored in data structures called filesystems
- A Linux system can have a practically unlimited number of filesystems
- These filesystems are self-contained and each has a logically hierarchical structure, using a file requires knowing where it is in the structure
- Every process that is running has a concept of where it "is" in the hierarchy of files - this is called the current working directory and it is used to allow us to type relative path names for files (a relative path specifies how to get to the file from "where we are")
- bash allows you to change your current working directory by using the command cd (e.g. cd somenewpath)

File Types

- Filesystems support holding data in multiple file types
  - regular files contain bytes of data
  - directory files contain lists of files
    - other types include symbolic links, devices, and various fifo mechanisms
- Directories can be created with the mkdir command (e.g. mkdir pathname) and removed with either the rm (e.g. rm -r pathname) or rmdir (e.g. rmdir pathname) command
- The list of files in a directory can be displayed using the ls command (e.g. ls pathname)

File Naming

- Files are identified using their names along with the name of the directory they are listed in - this is known as a path name
- File names are composed of any characters in your locale, but it is recommended to stick to upper case letters, lower case letters, numbers, -, _, .
- File names can be up to 255 characters long, path names can be up to 4096 characters long
- File names are generally case-sensitive in Linux
- File name restrictions such as these may be modified by the format of the filesystem created to hold them
- File and path names typed on the command line can be auto-completed by pressing Tab
- New empty files can be created using the touch command (e.g. touch pathname), removed using the rm command (e.g. rm pathname), and renamed using the mv command (e.g. mv oldpathname newpathname)

File Name Globbing

- In order to make it easier to enter multiple file names on command lines, bash supports a number of pattern matching characters which you can use like wildcards in file names - this is called globbing
  - * matches zero or more characters (e.g. a* h*k *th)
  - ? matches any one character (e.g. ? ?? ???* a?? b?d)
  - [ ] allows specifying a set of characters to match a single character in the file name (e.g. [abc] a[123] [A-Z]*)
- File names matching the pattern are placed on the command line in place of the pattern before the command runs

Shell Quoting and Escaping

- Since some characters on command lines have special meanings, we need a way to tell the shell when we want to use one of those without it meaning something special - we need a way of turning off, or escaping, the special meaning of these special characters

- \ will turn off the special meaning of whatever the next character is on the command line (e.g. * means the asterisk character, not file name globbing)

- ' ' turns off all special characters between the single quotes, except the single quote character (e.g. 'What the h* did you say!' turns off the special meaning of the * and ! characters and just makes them text)

- " " turns off the special meaning of most special characters with the notable exceptions of $ and " (e.g. "It's a *?" makes everything inside the double quotes just plain text with no special meanings)

Shell Command Invocation

- bash allows you to enter commands and start them running, then receive output from them, and know they finished
- If a program is going badly and you want to quit it immediately, press ^C (Control key and C simultaneously)
- Some commands are performed by bash itself with no outside help other than from the kernel, these are called built-in commands
- All other commands bash must find, stored in the filesystem somewhere, and start running on your behalf

Shell Command Types

- bash looks for commands in 3 places if they are not built-in:
  - defined as an alias (i.e. shortcuts or nicknames stored in bash's process memory)
  - in a function with the same name as the command entered (i.e. blocks of bash commands stored in bash's process memory)
  - in a file containing program code (the file must have the same name as the command entered and be located in a directory listed on bash's in-memory command path, and bash must have permission to run that file as a command)
- program code in files can be either binary code for the system processor, or a script
- There are commands to identify which type a command is, and where it comes from if it is stored somewhere (e.g. whereis echo, which echo, type echo)

Shell History

- bash keeps command history in memory and can save that to a file automatically
- You can recall commands from your history list using the up and down arrow keys on your keyboard
- You can use the right and left arrow keys to position your cursor on the current command line and begin typing wherever the cursor is, to edit the current command line
- There is also the older ! history access syntax which lets you recall commands or portions of commands from your history list directly

Shell Variables

- bash can store arbitrary named data using a storage mechanism called a variable
- A variable has a name and has data
- They are used to store various things including configuration information
- The variable HISTSIZE contains a number which tells bash how many commands to keep in the history list
- The variable PATH contains the list of directories to search for commands stored in files, a.k.a. the command path

Working With Variables

- You can set data into a variable (e.g. VARIABLENAME=data)
- You can delete a variable and its data (e.g. unset VARIABLENAME)
- You can see what is in a variable (e.g. echo $VARIABLENAME)
- Changing configuration variable contents changes the behaviour of commands that use those variables
- To have a variable always set to a desired value whenever you start a new bash shell, add a line to the end of the file .bashrc in your home directory setting that variable to that value

Lab 3
Command Line Interface

Logging in
Using SSH
Making and listing files
Globbing