Software Fundamentals Series

Workshop #3
Unix Basics

IEEE Computer Society Ryerson Chapter

October 2, 2018

What is Unix?

- Created in the late 1960s to early 1970s by AT&T-Bell labs as a new OS to be used internally

- Sold as an enterprise product and quickly became and industry standard

- Many different versions of Unix developed

- Led to the POSIX standard created by the IEEE Computer Society in 1988
    - *Portable Operating System Interface*
    - Defines a set of standards to allow programs to run across any OS based on Unix

GNU and Unix

- Project created in 1983 to develop an entirely free complete Unix-compatible software system
  - contained compilers, shell, text editors but not a core OS kernel

- The Linux Kernel project was started in 1991 by Linus Torvalds

- Linux gained popularity in the open source community and was eventually added as the official OS of GNU
  - Currently maintained by the Linux Foundation

- Today, there many different “flavours”/”distros” (distributions) of Linux that are used in all fields of software and hardware development

Why Learn Unix Systems?

- Unix-like systems are used EVERYWHERE
  - Mac OS and IOS are UNIX systems
  - Android OS is a distro of Linux
  - Almost all web servers in the world are run on Linux and Unix-like systems

- Almost all software companies use Unix systems for some area of development
  - Industry standard
  - Unix proficiency is a fundamental skill expected of any software developer/engineer

- Learning Unix systems makes you more **EMPLOYABLE!**

Command Line Interface (CLI)

- A *command line* interface (*CLI*) is a text-based user interface (UI) for the OS
    - The interface is commonly known as a “shell”

- Common CLIs
    - Bash (Bourne-again shell)
    - Bourne shell
    - Z shell
    - C shell
    - CMD (Command Prompt; Windows)

Unix CLI Tutorial

General Syntax and Info

- All commands and arguments are case sensitive
- Hit ENTER every time you want to run a command
- Basic command format
    - *command -optional flags arguments*
- Each command is its own program that may take arguments (such as file/directory names) and may be modifiable by a number of **flags**
    - **flags**: additional options specified by a - following the command name that modifies the operation of the command (most flags are common between commands but not all!)

IEEE computer society

IEEE Advancing Technology for Humanity

SSH

- *Secure Shell*

- Allows you to login to another remote machine

- General Syntax
    - ssh *username@domain*
    - *username* is an account set by the administrator
    - *domain* identifies the machine you logged into, it may be identified by a domain name or an IP

- You are now logged into an account on another machine, all operations that are performed are going to happen on the machine you logged into

First command example - ls and flags

- Basic syntax:
  - **ls**
  - Will list all directory and files in current directory

- Common flags usages
  - **ls -l** : shows all files/directories and permissions
  - **ls -a**: shows all files/directories including hidden files
  - Can combine flags (**applies to all commands)**
    - ls -la: show all files/directories and their permissions including hidden files
- Can use with arguments
  - **ls** *directory*: shows all files/directories within child directory(ies)
  - Can be combined with flags
    - **ls -la** *directory*

Directories

- To view current directory that you are in:
  - **pwd**
  - Will print out directory path you are in

- To change the current directory you are in
  - **cd** *path* (directories in path delimited by /)
  - **must use full path unless the directory you want to enter is a direct child of the current directory**

- Special Unix directory specifiers
  - **/** : “root”, the parent of all other directories
  - **..** : the parent of the current directory
  - **.** : the current directory
  - **~**: the “home” directory, the default directory that your account logs into

Moving Between Directories

- **cd .**
    - will not do anything since you are already in the current directory
- **cd ..**
    - will take you to the parent of the current directory
- **cd ../../**
    - will take you to the parent of the parent of the current directory (2 directories up)
- **cd ../***directory*
    - will take you to some other directory with in the parent
- **cd /**
    - will take you to the root directory, the parent of all other directories
- **cd** or **cd ~**
    - will take you to your home directory

Tips to Improve Efficiency

- Typing the first letters of a directory or filename and then hitting TAB will autocomplete the name of the directory/file you wish to type if it is present in path you are entering
  - If there are multiple file names with the same starting letters, hit TAB multiple times and it will either output the possibilities or cycle through them

- Hitting the UP directional key will cycle through the previous commands you've entered

- To recall a previous command based on the first letters
  - Type in first letters
  - Keep hitting CTRL+r until you find the command you want
  - Hit CTRL+e to select it

Tips to Improve Efficiency

- Wildcards can be used
  - The most common wildcard: *
  - Represents any number of characters in any combination
  - When used by itself, * = “all”
  - e.g **rm** * : will delete all files in current directory
  - e.g **rm** *.txt : will delete all files that end with .txt

- To kill a process
  - CTRL+c
  - Will stop any program or command currently running in the shell
  - Useful for refreshing the line if you made a mistake

mkdir

- Create directory
  - **mkdir** *directoryName*
  - **mkdir** *path/filename* : can create a directory in any path

touch

- Creates new, empty files
  - **touch** *filename*
  - **touch** *path/filename* : can create a file in any path

cp

- Copy file/directory to a new location

- For files
  - **cp** *filename pathToDesiredDirectory*

- For directories
  - Must use a flag
    - **-r** : recursive flag (**can be used for must commands that deal with directories)**
    - allows for the directory and all of its subdirectories to be manipulated
  - **cp -r** *directoryName pathToDesiredDirectory*

mv

- Move file/directory to a new location **and also used for renaming**

- Moving files/directories
  - **mv** *filename pathToDesiredDirectory*

- Renaming files/directories
  - **mv** *filename NewNonExistingName*
  - *Note: when moving directories, some shells may require the use of the -r flag or the -R flag

rm

- Remove file/directory

- Removing files
  - **rm** *filename*

- Removing directories
  - Must use **-r**
  - **rm -r** *directoryName*

- Common flag:
  - **-f** : represses messages, **works with most commands**
  - many shells will ask if you "really" want to delete each file, which can be tedious when deleting a large directory
  - **rm -rf** *directoryName* : **be careful when doing this**

CLI Text Editors

- Linux comes with a variety of command-line text editors
  - Essential for file editing on barebones Linux systems where a Graphical User Interface (GUI) is not available

- There are 3 most common editors:
  - *Vi/Vim*
  - *Nano*
  - *Emacs*

```
                :::
  iLE88Dj.  :jD88888Dj:
.LGitE888D.f8GjjjL8888E;
iE   :8888Et.     .G8888.
;i    E888,        ,8888,
      D888,        :8888:
      D888,        :8888:
      D888,        :8888:
      D888,        :8888:
      888W,        :8888:
      W88W,        :8888:
      W88W:        :8888:
      DGGD:        :8888:
                   :8888:
                   :W888:
                   :8888:
                    E888i
                    tW88D
```

ViM Editor

- One of the most common text editors
  - Comes default with many Linux distros

- Create/edit a file
  - **vim** *filename.extension*

- Vim has 3 modes: *normal mode*, *insert mode*, and *visual mode*
  - *Normal* (default, ‘*esc*’) - for commands like copy, delete, or indent
  - *Insert* (‘*i*’) - type to insert text
  - *Visual* (‘*v*’) - visual selection

- In *Normal* mode (can be combined):
  - **:w** - writes/saves the changes to the file
  - **:q** - quit vim (use **:q!** to not save changes)

wc

- ‘word count’
  - e.g. **wc** *filename.extension*

- Prints 5 column output, counting:
  - *newlines, words, characters, bytes, max. line length*

- Printing number of lines in a file
  - **wc -l** *test.txt*

- Common flags:
  - **-m** : number of characters
  - **-w** : number of words
  - **-c** : number of bytes

less

- Used to view the contents of a file
  - allows scroll through file one “page” at a time

- To view contents of a file
  - **less** *filename*
  - hit ‘q’ to exit view

cat

- Used for viewing the contents of multiple files at once
  - “concatenate”

- **cat** *file1 file2 file3 …*

- very useful when used with **redirection**

grep

- Used to search files for text

- **grep** “*pattern*” *fileordirectory*
  - will show you the matching patterns within the file/directory
  - commonly used with many flags
    - **-i** : case insensitive
    - **-w** : whole word
    - **-r** : recursive (for directories)
    - **-l** : list the files that contain the matching pattern

- can be used with regular expressions

Redirection

- Allows a file to be used as input/output instead of standard input/output
  - terminology
    - **standard output**: the shell (output appears in the shell when using standard output) by default
    - **standard input**: the keyboard, usually prompted by the shell by default
- The **>** symbol is used for output redirection
  - *command* **>** *filename*
  - e.g **ls >** *filename*
    - The output of **ls** will be stored in the specified file
    - **caution**: this will overwrite any existing file with the same specified name
    - **To Append to an existing file, use >>**
      - **ls >>** *existingFileName*

Redirection

- The **<** symbol is used for input redirection
  - e.g **cat <** *filename*
    - The input is taken from the file and fed to the command
    - Used with commands that usually require line-by-line input from the user
    - Not: the above example isn't very useful since it provides the same result as **cat** *filename*
    - **<** is more commonly used in complex strings of commands

Piping

- Connects the standard output of one command to the standard input of another (uses the output of one command as the input for another
- Uses the **|** symbol
  - *command1* **|** *command2*
  - e.g.
    - **ls | wc - l**
    - This will provide the line count resulting from the output of the **ls** command

Permissions

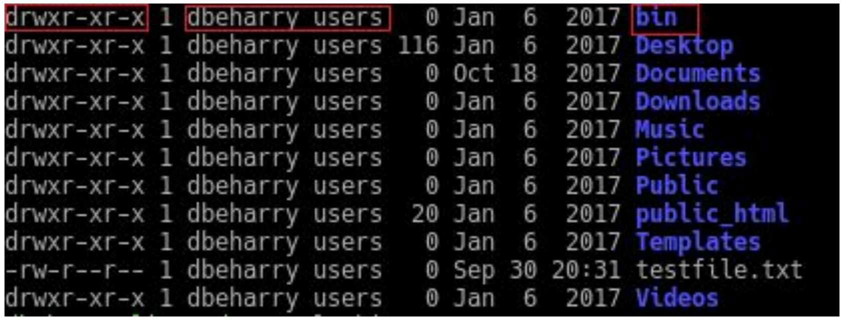
- Used to determine which users have access to which directories and files

- Usually divided into groups such as Users and SuperUsers
  - Super (aka Root) users usually have full permissions
  - If you are root user of a system, you can choose to use a non-root account for safety and use:
    - **su** : logs you in as a root user (prompts for password)
    - **sudo** *command* : use any command as root user once (prompts for password)

- Three kinds of permissions:
  - **Read**: the ability to view the contents of a file
  - **Write**: the ability to modify the contents of a file
  - **Execute**: the ability to run a program

Permissions

- Permission representation in Unix
- e.g ls -l



```
drwxr-xr-x 1 dbeharry users   0 Jan  6  2017 bin
drwxr-xr-x 1 dbeharry users 116 Jan  6  2017 Desktop
drwxr-xr-x 1 dbeharry users   0 Oct 18  2017 Documents
drwxr-xr-x 1 dbeharry users   0 Jan  6  2017 Downloads
drwxr-xr-x 1 dbeharry users   0 Jan  6  2017 Music
drwxr-xr-x 1 dbeharry users   0 Jan  6  2017 Pictures
drwxr-xr-x 1 dbeharry users   0 Jan  6  2017 Public
drwxr-xr-x 1 dbeharry users  20 Jan  6  2017 public_html
drwxr-xr-x 1 dbeharry users   0 Jan  6  2017 Templates
-rw-r--r-- 1 dbeharry users   0 Sep 30 20:31 testfile.txt
drwxr-xr-x 1 dbeharry users   0 Jan  6  2017 Videos
```

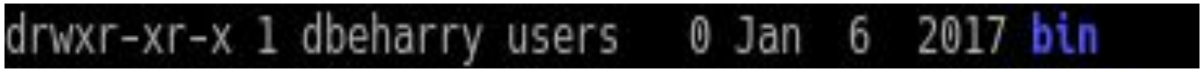
- 4 columns

Object Type	Current User's Permissions	Permissions granted to the rest of the User's Group	Permissions granted to all other users
either: d :directory - :file	3 characters, representing read, write, execute in that order	3 characters, representing read, write, execute in that order	3 characters, representing read, write, execute in that order

Permissions

- Permissions columns

Read	Write	Execute
r = has read access	w = has write access	x = has execute access
- = doesn't have read access	- = doesn't have write access	- = doesn't have execute access

- E.g 

```
drwxr-xr-x 1 dbeharry users     0 Jan  6  2017 bin
```

- Indicates that bin is a directory
- rwx: User dbeharry has read write and execute access to bin
- r-x: Other users in the "users" group have only read and execute access to bin
- r-x: all other users outside of the "users" group have only read and execute access to bin

Permissions

- Permissions can be changed using the **chmod** command
  - E.g. **chmod** *permissionsCode filename*
  - the permissions code is determined by the binary representation of the desired combination of rwx

- Please see resources for more details

Variables

- Variables can be used for a variety of reasons, **especially in shell scripts**
  - *variableName = variableContent*

- bash uses variables in the background for a variety of reasons
- Common variables are:
  - **PATH** : a list of directories that allows programs to be executed from anywhere
  - **PWD** : the current directory you are in
- To view the contents of a variable, use the **echo** command and the **$** operator
  - e.g. **echo $PATH** : prints out the contents of PATH
  - The **$** operator in front of a variable accesses the variable's content
  - **echo** *text* : simply prints out text to standard output
  -

Variables

- When declared, variables only exists for the current session
  - They are **local**
  - To be persistent and **global**
    - **export** and/or addition to **.bashrc**

- Please see attached info for more details

...And MUCH More

- Shell scripts, aliases, bash configs and MUCH, MUCH more

- There is a lot to learn in the Unix environment! Practice as often as possible to get used to shell!

- Index of Bash commands
  - https://ss64.com/bash/

- Variables and shell scripts ---**VERY USEFUL**
  - https://www.tutorialspoint.com/unix/unix-using-variables.htm
  - https://www.shellscript.sh/

- bash configuration and .bashrc
  - http://www.hypexr.org/bash_tutorial.php#config
-
- In-depth Vim tutorial
  - https://www.tutorialspoint.com/vim/index.htm