Linux CLI

Concepts

OS

- OS: operating system
- Examples
  - Microsoft Windows
  - Mac OS (based on Linux)
  - Linux
  - iPhone iOS (based on Linux)
  - Android (based on Linux)

Layering

- OSs are layered
  - Kernel is the innermost layer
  - Desktop GUI (graphical user interface) is outermost layer
- Peeling off layers
  - Removing outer layers that we don’t need
  - Linux can
  - Most other OSs (Windows, Mac, iOS, Android) cannot

Peeling Off Layers in Linux

- Desktop GUI layer can put 50% or even higher load on the OS
- Removing the desktop GUI can double or more our application performance
- Useful
  - Cloud
  - VMs
  - Containers
  - IoT (Internet of things)
  - Embedded systems

Desktop GUI vs. CLI

- Desktop GUI
  - Easy to use
  - Adds tremendous overhead—loss of performance
  - Not scriptable
- CLI (command line interface)
  - Hard to use
  - Little to no overhead—no loss of performance
  - Highly scriptable

Non-desktop GUIs

- Workarounds for not having a desktop GUI
  - Small web server, user accesses via desktop web browser
  - GUI app on Windows or Mac that connects to Linux VMs
- Adds little to no overhead
- Not as nice as desktop GUI, but usually adequate when we cannot afford the overhead of a desktop GUI

Labs

Linux CLI

- Connecting, logging in
- Users, groups, permissions
- Files and directories
  - Creating, deleting, copying, moving, setting permissions
- Processes and threads
  - Creating, killing, scheduling, stack, heap

The End

Linux CLI

Business Cases

Major Linux Distros (Distributions)

GNU (recursive acronym, GNUs not Unix) Linux

- Red Hat
  - CentOS
    - Amazon Linux
    - Oracle Linux
  - Fedora
- Debian
  - Ubuntu
    - Lubuntu
  - Raspberry Pi OS—formerly Raspbian
  - Various IoT

Red Hat Branch

Red Hat: stable, large, corporate, servers, paid

- CentOS: free version of Red Hat
  - Amazon Linux
  - Oracle Linux
- Fedora: experimental, new features for Red Hat tested

Debian Branch

Debian: stable, lean, mean

- Ubuntu: Debian plus packages and drivers, desktop
  - Lubuntu: pared down, old computers, desktop VMs
- Raspberry Pi OS—formerly Raspbian
- Various IoT

Administrative Differences

- Debian and Red Hat differ in administrative
  - Install, bootstrap, shutdown
  - Package managers
  - Patches, updates
  - Security
- Within the Debian or Red Hat branches, very little differences

Fortune 500 Company Servers

- Servers: Red Hat
  - Stability
  - Support
  - Consulting
  - Training
  - Auditable
- Desktop: typically would run Windows

Linux on Desktop

- Ubuntu
  - Has drivers for most devices: cameras, printers, etc.
  - Red Hat: drivers for keyboard and mouse, not much else
  - Target market
- Consider Lubuntu if the hardware is low-end

Old Computer

Lubuntu

- Pared down version of Ubuntu
- Drivers available

Linux VM With GUI

Lubuntu

- Pared down version of Ubuntu

Academic Research

- Debian on servers
- Ubuntu on desktop
  - Consider Lubuntu on desktops depending on how beefy the hardware is
- Lubuntu for VMs that need a desktop GUI

IoT Device

Debian

- Stable, lean, mean
- Always add any needed packages

The End

BASH Shell

Concepts

Linux Shells

- Program that creates a Linux CLI
- Several shells: Bourne, C, Korn, T, Z, BASH, etc.
- BASH (Bourne-again shell)
  - Most widely-used shell in the Linux environment
  - De facto standard

Shell Scripts

- Scripts that allows us to programmatically run Linux CLI commands
- Programming constructs
  - Variables
  - If statements
  - Loops
  - Etc.

Lab

Basics of BASH shell programming

The End

BASH Shell

Business Cases

Installation

Create BASH shell scripts to install software

Production Jobs

- Create a BASH shell script for each production job
- Use a job scheduler to run scripts
  - Specify dependencies
    - Time
    - Other jobs
    - File arrival
    - Other triggers

Common Repetitive Tasks

- Create BASH shell script for common repetitive tasks
- Run the script ad hoc when the need arises

The End

Github Git CLI

Concepts

Without Source Code Control

- Multiple programmers working on the same software
  - One programmer overwrites another programmer’s changes
- No separation between production code and development code
  - Programmers can put untested code into production
- No versioning of software
  - No way to do releases by version number

Source Code Control

Allows:

- Multiple programmers working on the same software coordinating changes and merges
- Production code to be separate from QA (quality assurance) and development code
- Elaborate versioning for releases
  - Examples: 8.1.4, 9.2.1, 10.3.7

Lab

- Since we will be working in the cloud without a GUI, we will need to learn to use the GitHub git CLI.
- For this course, we will need to:

1. Clone a repo
2. Create a branch
3. Change to a branch
4. Sync a branch
5. Track changes
6. Stage changes
7. Commit changes
8. Push a branch to GitHub

The End

Github Git CLI

Business Cases

Software Vendor

- Current major release is 10.4.1
  - Customers still on 10.3.6, 10.3.7, 10.4.0
- Previous major release is 9.7.2
  - Customers still on 9.6.4, 9.6.5, 9.70, 9.71
- Just released 11.0.0-beta
- Developing 11.0.1-beta

Alpha/Beta

- Alpha release: final internal release prior to beta release
- Beta release: initial external release to limited number of customers who are willing

Bug Found

- We must fix: 9.6.4, 9.6.5, 9.7.0, 9.7.1, 9.7.2, 10.3.6, 10.3.7,10.4.0, 10.4.1, 11.0.0-beta, 11.0.1-beta
- Using source code control, we have branches for all of these
- For each branch
  - Make a new branch
  - Fix the bug
  - Test
  - Merge
  - Release

The End