A Brief Introduction to Linux [*]

David Wright

February 20 2021

What We'll Be Covering Today

1. History of Linux
2. Anatomy of a Linux system
3. Introduction to the shell and command line + some demos

History of Linux

The Road to Operating Systems

- Computers as we know them have their roots in the 1940s
  - Electronic Numerical Integrator and Computer (ENIAC) - University of Pennsylvania (1945)
  - Colossus - British Military (1943)
- Computers finally get memory in 1949
  - EDVAC (Electronic Discrete Variable Automatic Computer) and EDSAC (Electronic Delay Storage Automatic Calculator)
  - Data was represented as waves in mercury filled tubes

- ENIAC and Colossus had no memory
  - ENIAC was hard wired for each program
  - Colossus used paper tapes
- EDVAC was the successor to ENIAC and EDSAC was a project out of Cambridge University
- The mercury tubes worked via transducers, could cycle data back into tube or read it

ENIAC and Mercury Delay Line Memory

[*] This PDF document is an inferior version of an OER HTML page; free/libre Org mode source repository.

The Road to Operating Systems (cont)

- IBM releases the IBM 701 in 1952
  - First true assembly code and reusable code
- The UNIVAC 1103A introduces the **interrupt**, allowing a processor to switch between jobs
- IBM created SHARE (Society to Help Alleviate Redundant Effort), an IBM user organization, to maintain common routines
- Prior to UNIVAC, computers only ran 1 program at a time

Operating Systems

- In the 1950s, batch processing "operating systems" came onto the scene
- In 1954, FORTRAN was released
  - With high-level languages, programmers didn't have to know about the architecture of a computer
- In the 1964, Multics was released
  - Hierarchical file system
  - Written in a high level language
  - Filesystem security, and more!
- Multics was big and bloated, so Bell Labs created UNIX in the late 60s
  - Computers of old could only run one program at a time
    - Batch processing operating systems (very basic) allowed computers to run batches of jobs sequentially
  - In the 60s we saw computers get smaller, cheaper, and easier to use
    - multiprogamming and multiprocessing became more popular
  - Multics was a turning point in the history of OS's
  - UNIX is a play on Multics (UN vs Mul because it is simpler)

Operating Systems (cont.)

- In 1973, UNIX 4th edition was released
  - Written in C which made it easy to recompile for different architectures
- In the 70s and 80s, we saw the arrival of Windows and OSX
  - Academics and Researchers still use UNIX
- In 1984, the Bell Labs system was broken up
  - Now AT&T, they sought to get into the computer business and revoked the free licensing of UNIX to universities
- Minix, a UNIX-like operating system, was created soon after but was only freely available to universities and researchers

Operating Systems (cont.)

- In 1991, Linus Torvalds released Linux
  - Linux was UNIX-like, and was completely free (speech and beer)
  - It saw quick adoption by previous researchers who used UNIX
  - The open-source development of Linux allowed it to progress rapidly

Anatomy of a Linux System

First, what is Linux?

- It is just a kernel. It manages the following
  - System Memory
  - Software programs
  - Hardware
  - File system
- It needs basic programs in order to be a complete operating system
  - Historically, it has bundled the GNU coreutils

Four Basic Parts of a Linux System

- The Linux kernel
- The GNU utilities
- A graphical desktop environment
- Application software

The GNU Utilities

- GNU (GNU's Not UNIX) organization developed a complete set of Unix utilities for:
  - Handling files
  - Manipulating text
  - Managing processes
- They had no kernel to run them on until they started getting bundled with Linux

System Memory Management

- The kernel doesn't only manage physical memory
  - It can also create and manage virtual memory somewhere on the disk called the **swap space**
- The kernel **swaps** memory locations back and forth from physical memory to swap space
- Memory locations are grouped into **pages**
  - The kernel maintains a table with page locations (swap or physical)
  - The kernel **swaps out** pages that have not been access for a period of time

System Memory Management (cont)

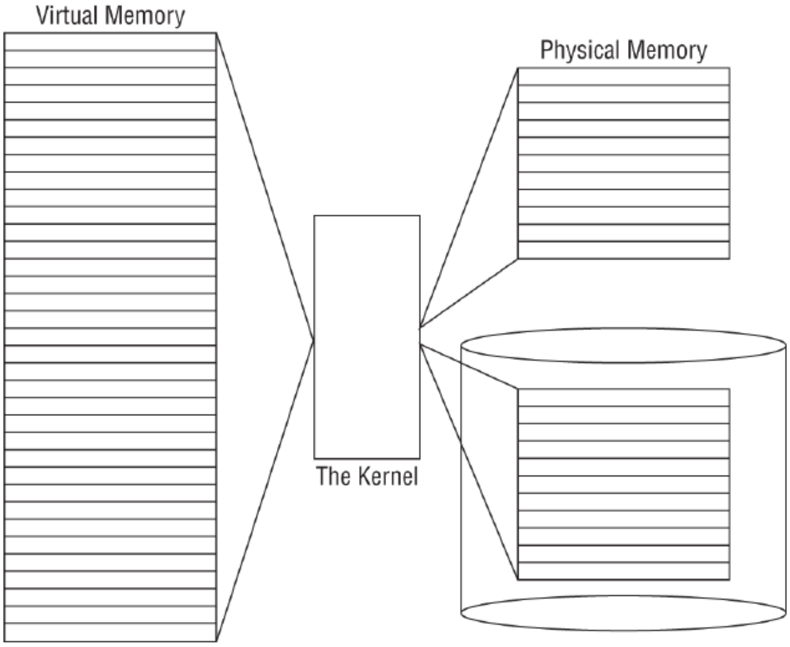


Figure 6: Diagram showing the virtual memory, physical memory, and table of memory pages

Program Management

- A running program in Linux is a **process**
- The kernel creates the **init** (first) process which starts all other processes

- **Systemd** is the most popular Linux initialization and process management system. It can start processes when:
  - the system boots
  - a particular hardware device is connected
  - a service is started
  - a network connection is established
  - a timer has expired

More Systemd

- **unit files** are linked to events and determine what processes to run
- **targets** are groups of unit files that define a specific state of the system
- Example: At startup, the `default.target` unit defines all the unit files to start.

Hardware Management

- The kernel needs **driver code** to know how to drive a particular device
- In the past, the only way to add the driver code was to recompile the kernel with it added
- Kernel modules allow us to insert driver code into a running kernel without having to recompile

Types of Hardware Devices

- Character
  - Devices that can handle one character at a time, such as modems and terminals
- Block
  - Devices that can handle large **blocks** of data, such as drives
- Network
  - Devices that use packets to send and receive data, such as network cards

Interaction with Devices

- Linux creates special files called **nodes** for each device
- All communication is done through the device node
- Each node has a unique number pair that identifies it to the kernel
  - Major number: similar devices are grouped with this number
  - Minor number: identifies specific device in major group

File Systems

- The Linux kernel supports many file systems
- The kernel interacts with each file system using the Virtual File System (VFS)
  - Provides a standard interface for kernel to file system communication

The Shell

- An interactive utility that you interact with via the command line
- Allows you to start programs, manage files, manage processes, etc.
- You can group shell commands together into files to execute as a program
- A few shells are available, but the Bash shell is most common
- Break the constraints of a GUI
  - String together multiple commands using **pipes** (|) and create a **pipeline**

Graphical Environments on Linux

- In the early 90s, only text interfaces were available
- Now, the X Window software allows Linux to use graphical interfaces
- The two main packages that provide the X Window software are
  - X.org (older, more mature)
  - Wayland (newer, more secure, easier to maintain)
- The X Window software by itself only produces a graphical display environment for individual applications
  - If you want one of the now standard desktop environments (GNOME, KDE), you'll need to install it separately

The Linux File System Hierarchy Standard (FHS)

Before we go into demos, let's learn a little about the Linux file system

- As opposed to Windows, Linux doesn't have "C" or "D" drives
- All disks are mounted under the **root** ("/") , a single base directory in what's called the "virtual directory"

Common Linux Directories

- / Root of the virtual directory, usually no files are placed here (only other directories)
- **/boot** Directory where boot files are stored
- **/dev** Where Linux creates device nodes
- **/etc** System configurations
- **/home** User directories
- **/media** Common place to mount external drives
- **/tmp** A special directory, only holds files temporarily
- **/usr** Many things go here, but it is most often used for user-installed programs

Shell and Command Line + Demos

The Shell Prompt

- In the upper left we have the **prompt**
  - `user@host`
  - Also shows the current directory
- When you first log in, you'll be dropped into your home directory (˜)

[width=.9]figures/dave/01-login

Navigating the File System

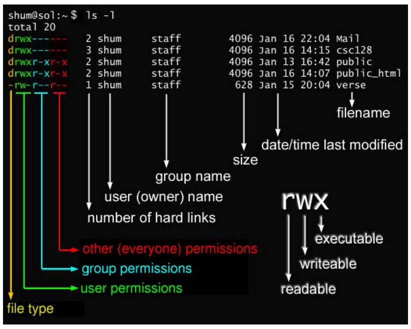
- **pwd** prints the working directory (where you are)
- **cd** changes directories
  - If ran without any arguments, it takes you to your home directory
  - Can use absolute (starting at the root) or relative paths
  - Can use **..** to reference the parent directory
  - As we move around, the prompt reflects the current directory

[width=.9]figures/dave/02-nav

Navigating the File System (cont)

- **ls** lists the contents of a directory
  - **ls -l** gives a long listing with better structure and more information (permissions, file vs directory, etc.)
  - **ls -a** lists all files, even dot-files
  - **ls -la** combines the **-l** and **-a** options

[width=.9]figures/dave/02-nav



```
shum@sol:~$ ls -l
total 20
drwx------  2 shum     staff         4096 Jan 16 22:04 Mail
drwx------  3 shum     staff         4096 Jan 16 14:15 csc128
drwxr-xr-x  2 shum     staff         4096 Jan 13 16:42 public
drwxr-xr-x  2 shum     staff         4096 Jan 16 14:07 public_html
-rw-r--r--  1 shum     staff          628 Jan 15 20:04 verse
```

Using the Manual

- The **man** command lists the manual for a given command
- If you don't know the specific name, use the **-k** option to search by keyword
- You can even **man man**

[width=.9]figures/dave/03-man

Moving and Copying Files

- The **mv** and **cp** commands move and copy files
- **mv** doesn't move data (if in same file system)
  - Directory entries just get updated
  - **mv** can move directories, **cp -R** can copy directories

[width=.9]figures/dave/04-mvcp

- The **-i** option prevents you from overwriting existing files

Creating and Removing Files

- The **rm** command removes files
  - **rm is forever**, don't forget it
- The **touch** command creates an empty file

[width=.9]figures/dave/05-rmtouch

- The **-i** option prompts you when removing files

Creating and Removing Directories

- **mkdir** makes directories
  - **mkdir -p** can create nested directories
- **rmdir** removes empty directories
  - **rm -r** will remove directories and their contents, but **be careful**

[width=.9]figures/dave/06-dirs

Viewing File Contents

- **cat** will output all of the file contents to the screen
- **less** is a pager. It will let you scroll through your content
- **tail** and **head** show you the end or beginning of your file
  - the **-n** option lets you specify the number of lines to show

[width=.9]figures/dave/07-cat

Editing File Contents

- **sed** (Stream EDitor) is a powerful command line tool for modifying files
  - In the example, I use it to replace all occurrences of “Hello” with “Goodbye”
- There are also multiple command line text editors
  - **nano** is a very basic text editor that is included with most Linux distributions

[width=.9]figures/dave/08-ed

Editing File Permissions

- We often need to change the permissions on a file
- **chmod** (change mode) allows us to tweak file permissions
  - In the example, I give only my user execute privileges (**u+x**) on “hello-world.py”

[width=.9]figures/dave/09-chmod

Searching Files and File Globbing

- **grep** lets you search the contents of files (and more)
  - The **-i** option is for case insensitive searches
  - The **-v** option finds the lines which don't have the search
  - The **-n** option gives line numbers
- **find** helps you search for files
  - I use **.** to search the current directory and the **-name** option to search by file name

[width=.9]figures/dave/10-grfd

- I also introduced file globbing via wildcards (not an exhaustive example of wildcards)
  - The **?** represents any single character
  - The **[ ]** specify a range
  - The ***** matches anything. I use it to find the only `.py` file

Output Redirection and Pipelines

- The right arrow > can be used to redirect the output of a command
  - Notice that a single arrow overwrites the file
  - A double arrow » appends
- In the example, I use a pipe | to use the output of the **cat** command as the input to the **less** command

[width=.9]figures/dave/11-pipe

The End

That about wraps up what I can reasonably cover in an intro lecture. Please try these examples out on your own, and maybe try something new as well!

Further Reading

- *Linux Command Line and Shell Scripting Bible*
  - `https://bit.ly/3k7Zy1m` (UCF Library)
- `https://linuxjourney.com/`