Week-4 Reading Material

Introduction to Linux

Why Linux?

Why should you be interested in Linux? If you look around our society, you primarily see people using Windows, Macintosh OS X, Linux, and Unix in one form or another. Below, we see an estimated breakdown of the popularity of these operating systems as used on desktop computers.
Note: Estimates vary by source. These are as of July 2019.

- Microsoft Windows: 77.6%
- Macintosh (Mac OS X): 13.2%
- Linux or Unix: 1.6% (There are numerous distributions or "flavors" of Linux and Unix)

If we expand beyond desktop computers, we can find other operating systems which run on mainframe computers, minicomputers, servers, and supercomputers. Some of the operating systems for mainframe/minicomputer/server/supercomputer are hardware-specific such as IBM System i (initially AS/200) for IBM Power Systems. For mobile devices, there are still a wide variety of systems but the most popular two are iOS (a subset of Mac OS X) and Android (Linux-based).

With such a small market share for Linux/Unix, why should we bother to learn it? Certainly, for your career you will want to learn about the more popular operating system(s). Well, there are several reasons to learn Linux/Unix.

First, the Macintosh operating system (Mac OS X) runs on top of a Unix operating system (based on the Mach Unix kernel). Thus, the real market share is larger than it first appears. Additionally, many of the handheld computing devices run a version of Linux (e.g., Google Android). In fact, estimates are that Android-based devices make up at least 20% of the handheld device market. There are other hardware devices that also run Linux or Unix such as firewalls, routers, and WiFi access points. So, Linux, Unix, Mac OS X, and the operating systems of many handheld devices comes to more than 15% of the total operating system usage.

Second, and more significant, is the fact that a majority of the servers that run on the Internet run Linux or Unix. In 2011, a survey of the top 1 million web servers indicated that nearly 2/3s ran Unix/Linux while slightly more than 1/3 ran Microsoft Windows. The usage of Unix/Linux is not limited to web servers as the platform is popular for file servers, mail servers, database servers, and domain name system servers.

That still might not explain why you should study Linux. So, consider the following points.

Open source (free)—The intention of open source software is to make the source code available. This gives programmers the ability to add to, enhance or alter the code and make the new code available. It is not just the Linux operating system that is open source but most of its application software. Enhancements can help secure software so that it has fewer security holes. Additions to the software provide new features. Alterations to software give users choices as to which version they want to use. Linux, and all open source software, continues to grow at an impressive rate as more and more individuals and groups of programmers contribute. As a side effect, most open source software is also freely available. This is not true of all versions of Linux (e.g., Red Hat Enterprise Linux is a commercial product) but it is true of most versions.

Greater control—Although Windows permits a DOS shell so that the user can enter commands, the DOS commands are limited in scope. In Linux/Unix, the command line is where the power comes from. As a user, you can specify a wide variety of options in the commands entered via the command line and thus control the operating system with greater precision. Many users become frustrated with the Windows approach where most control is removed from the users' hands. The Windows approach is one that helps the naive user who is blissfully unaware of how the operating system works. In Linux, the user has the option to use the GUI but for those who want to be able to more directly control the operating system rather than be controlled by the operating system, the command line gives that level of access.

Learning about operating systems—As stated in the last paragraph, Windows helps shelter users from the details of what the operating system is doing. This is fine for most users who are using a computer to accomplish some set of tasks (or perhaps are being entertained). But some users will desire to learn more about the computer. With the command line interface (CLI) in Unix/Linux, you have no choice but to learn because that is the only way to know how to use the command line. And learning is fairly easy with the types of support available (e.g., man pages).

The Linux Operating System: GUIs

As with most operating system today, Linux can be operated through a GUI. Many users only have experience with a GUI and may not even be aware that there is a text-based (command-line) interface available. It is assumed, because you are taking this course, that you want to understand the computer's operating system at more than a cursory level. We will want to explore beyond the GUI. In Linux, it is very common to use the CLI as often, or even more often, than the GUI. This is especially true of a system administrator because many tasks can only be performed through the CLI.

1.4 The Linux Command Line

As most of the rest of the text examines the CLI, we only present a brief introduction here. We will see far greater detail in later chapters.

1.4.1 The Interpreter

The CLI is part of a shell. The Linux shell itself contains the CLI, an interpreter, and an environment of previously defined entities like functions and variables. The interpreter is a program which accepts user input, interprets the command entered, and executes it.

The interpreter was first provided as a mechanism for programming. The programmer would enter one program instruction (command) at a time. The interpreter takes each instruction, translates it into an executable statement, and executes it. Thus, the programmer writes the program in a piecemeal fashion. The advantage to this approach is that the programmer can experiment while coding. The more traditional view of programming is to write an entire program and then run a compiler to translate the program into an executable program. The executable program can then be executed by a user at a later time.

The main disadvantage of using an interpreted programming language is that the translation task can be time consuming. By writing the program all at once, the time to compile (translate) the program is paid for by the programmer all in advance of the user using the program. Now, all the user has to do is run the executable program; there is no translation time for the user. If the program is interpreted instead, then the translation task occurs at run-time every time the user wants to run the program. This makes the interpreted approach far less efficient.

While this disadvantage is significant when it comes to running applications software, it is of little importance when issuing operating system commands from the command line. Since the user is issuing one command at a time (rather than thousands to millions of program instructions), the translation time is almost irrelevant (translation can take place fairly quickly compared to the time it takes the user to type in the command). So, the price of interpretation is immaterial. Also consider that a program is written wholly while your interaction with the operating system will most likely vary depending upon the results of previous instructions. That is, while you use your computer to accomplish some task, you may not know in advance every instruction that you will want to execute but instead will base your next instruction on feedback from your previous instruction(s). Therefore, a compiled approach is not appropriate for operating system interaction.

1.4.2 The Shell

The interpreter runs in an environment consisting of previously defined terms that have been entered by the user during the current session. These definitions include instructions, command shortcuts (called aliases), and values stored in variables. Thus, as commands are entered, the user can call upon previous items still available in the interpreter's environment. This simplifies the user's interactions with the operating system.

The combination of the interpreter, command line, and environment make up what is known as a shell. In effect, a shell is an interface between the user and the core components of the operating system, known as the kernel (refer back to Figure 1.2). The word shell indicates that this interface is a layer outside of, or above, the kernel. We might also view the term shell to mean that the environment is initially empty (or partially empty) and the user is able to fill it in with their own components. As we will explore, the components that a user can define can be made by the user at the command line but can also be defined in separate files known as scripts.

The Linux environment contains many different definitions that a user might find useful. First, commands are stored in a history list so that the user can recall previous commands to execute again or examine previous commands to help write new commands. Second, users may define aliases which are shortcuts that permit the user to specify a complicated instruction in a shortened way. Third, both software and the user can define variables to store the results of previous instructions. Fourth, functions can be defined and called upon. Not only can commands, aliases, variables and functions be entered from the command line, they can also be placed inside scripts. A script can then be executed from the command line as desired.

The Linux shell is not limited to the interpreter and environment. In addition, most Linux shells offer a variety of shortcuts that support user interaction. For instance, some shells offer command line editing features. These range from keystrokes that move the cursor and edit characters of the current command to tab completion so that a partially specified filename or directory can be completed. There are shortcut notations for such things as the user's home directory or the parent directory. There are also wild card characters that can be used to express "all files" or "files whose names contain these letters," and so on. See Figure 1.16 which illustrates the entities that make up a shell.

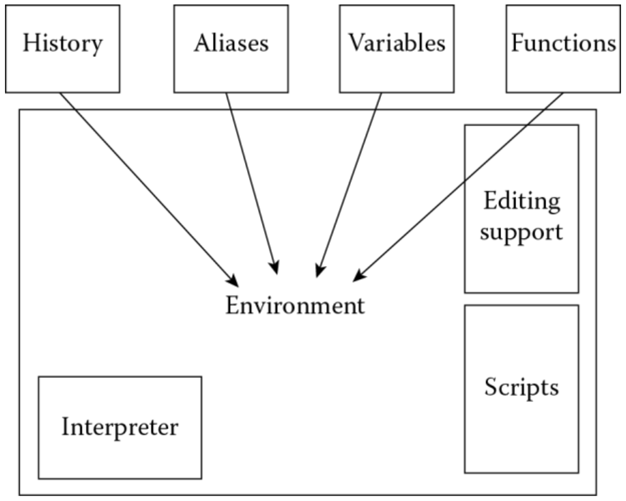


Figure 1.16: Components of a Linux shell.

A shell is provided whenever a user opens a terminal window in the Linux GUI. The shell is also provided when a user logs into a Linux system which only offers a text-based interface. This is the case if the Linux system is not running a GUI, or the user chooses the text-based environment, or the user remotely logs into a Linux system using a program like telnet or ssh. Whichever is the case, the user is placed inside of a shell. The shell initializes itself which includes some predefined components as specified by initialization scripts. Once initialized, the command line prompt is provided to the user and the user can now begin interacting with the shell. The user enters commands and views the responses. The user has the ability to add to the shell's components through aliases, variables, commands, functions, and so forth.

1.4.3 The CLI over the GUI

As a new user of Linux, you might wonder why you should ever want to use the CLI over the GUI. The GUI is simpler, requires less typing, and is far more familiar to most users than the command line. The primary difference between using the GUI and the CLI is a matter of convenience (ease) and control.

In favor of the GUI:

- *Less to learn*: Controlling the GUI is far easier as the user can easily memorize movements; commands entered via the command line are often very challenging to remember as their names are not necessarily descriptive of what they do, and many commands permit complex combinations of options and parameters.
- *Intuitiveness*: The idea of moving things by dragging, selecting items by pointing and clicking, and scrolling across open windows becomes second nature once the user has learned the motions; commands can have archaic syntax and cryptic names and options.
- *Visualization*: As humans are far more comfortable with pictures and spatial reasoning than we are at reading text, the GUI offers a format that is easier for us to interpret and interact with.
- *Fun*: Controlling the computer through a mouse makes the interaction more like a game and takes away from the tedium of typing commands.
- *Multitasking*: Users tend to open multiple windows and work between them. Because our computers are fast enough and our operating systems support this, multitasking gives the user a great deal of power to accomplish numerous tasks concurrently (i.e., moving between tasks without necessarily finishing any one task before going on to another). This also permits operations like copying and pasting from one window to another. Although multitasking is possible from the command line, it is not as easy or convenient.

In favor of the command line:

- *Control*: Controlling the full capabilities of the operating system or applications software from the GUI can be a challenge. Often, commands are deeply hidden underneath menus and pop-up windows. Most Linux commands have a number of options that can be specified via the command line to tailor the command to the specific needs of the user. Some of the Linux options are not available through the GUI.
- *Speed*: Unless you are a poor typist, entering commands from the command line can be done rapidly, especially in a shell-like Bash which offers numerous shortcuts. And in many instances, the GUI can become cumbersome because of many repetitive clicking and dragging operations. On the other hand, launching a GUI program will usually take far more time than launching a text-based program. If you are an impatient user, you might prefer the command line for these reasons.
- *Resources*: Most GUI programs are resource hogs. They tend to be far larger in size than similar text-based programs, taking up more space on hard disk and in virtual memory. Further, they are far more computationally expensive because the graphics routines take more processing time and power than text-based commands.
- *Wrist strain*: Extensive misuse of the keyboard can lead to carpal tunnel syndrome and other health-related issues. However, the mouse is also a leading cause of carpal tunnel syndrome. Worse still is constant movement between the two devices. By leaving your hands on the keyboard, you can position them in a way that will reduce wrist strain. Keyboards today are often ergonomically arranged and a wrist rest on a keyboard stand can also reduce the strain.
- *Learning*: You can learn a lot about operating systems by exploring Linux. However, to explore Linux, you mostly do so from the command line. By using the command line, it provides you with knowledge that you would not gain from GUI interaction.

The biggest argument in favor of the command line is control. As you learn to control Linux through the command line, hopefully you will learn to love it. That is not to say that you would forego the use of the GUI. Most software is more pleasing when it is controlled through the GUI. But when it comes to systems-level work, you should turn to the command line first and often.

Unix and Linux

Unix is an old operating system, dating back to 1969. Its earliest incarnation, known as MULTICS, was developed for a single platform. It was developed by AT&T Bell Labs. Two of the employees, Dennis Ritchie and Ken Thompson, wanted to revise MULTICS to run as a platform-independent operating system. They called their new system Unics, with its first version being written in the assembly language of the DEC PDP-11 computer so that it was not platform-independent. They rewrote the operating system in the C programming language (which Ritchie developed in part for Unix) to make it platform independent. This version they named Unix.

Numerous versions of Unix were released between 1972 and the early 1980s including a version that would run on Intel 8086-based computers such as the early IBM PC and PC-compatible computers. Unix was not a free operating system. In spite of it being implemented as a platform-independent operating system, it was not available for all hardware platforms.

In 1983, Richard Stallman of MIT began the GNU Project, an effort to complete a Unix-like operating system that was both free and open source. GNU stands for GNU's Not Unix, an indication that GNU would be a Unix-like operating system but separate from Unix. His goal was to have anyone and everyone contribute to the project. He received help from programmers around the world who freely contributed to the GNU operating system, which they wrote from scratch. Although a completed version of GNU was never released, the approach taken was to lead to what we now call the open source community. Stallman formed the Free Software Foundation (FSF) and defined the GNUs General Public License (GPL).

At around the same time as the initiation of the GNU Project, researchers at the University of California Berkeley developed their own version of Unix, which was given the name BSD (Berkeley Standard Distribution) Unix. This version includes networking code to support TCP/IP so that these Unix computers could easily access the growing Internet. In time, BSD 4.2 would become one of the most widely distributed versions of Unix.

The result of several competing forms of Unix led to what some have called the "Unix Wars." The war itself was not restricted to fighting over greater distribution. In 1992, Unix System Laboratories (USL) filed a lawsuit against Berkeley Software Design, Inc and the Regents of the University of California. The lawsuit claimed that BSD Unix was built, at least partially, on source code from AT&T's Unix, in violation of a software license that UC Berkeley had been given when they acquired the software from AT&T. The case was settled out of court in 1993.

By 1990, the open software foundation, members of the open source community had developed standardized versions of Unix based on BSD Unix. Today, there are still many different distributions of Unix available which run on mainframe, minicomputers, and servers.

In 1991, a student from Finland, Linus Torvalds, was dissatisfied with an experimental operating system that was made available through an operating systems textbook of Andrew Tanenbaum. The operating system was called Minix. It was a scaled down Unix-like operating system that was used for educational purposes. Torvalds decided to build his own operating system kernel and provide it as source code for others to play with and build upon. Early on, his intention was just to explore operating systems. Surprisingly, many programmers were intrigued with the beginnings of this operating system, and through the open source community, the operating system grew and grew.

The development of Linux in many ways accomplished what Stallman set out to do with the GNU project. Stallman and many in the FSF refer to Linux as GNU/Linux as they claim that much of Linux was built on top of the GNU project code that had been developed years earlier.

According to some surveys, roughly 75% of Linux has been developed by programmers who work for companies that are investing in Linux. The GPL causes many of these programmers to publish their code rather than keeping the code proprietary for the companies they work for. Additionally, 18% of the code is developed strictly by volunteers who are eager to see Linux grow.

Today, Linux stands on its own as a different operating system from Unix. In many cases, Linux is freely available in source code and the open source community continues to contribute to it. And like Unix, there are many distributions. Unlike Unix, however, Linux' popularity is far greater because, while Unix can run on personal computers, Linux is geared to run on any platform and is very effective on personal computers.

Linux Distributions

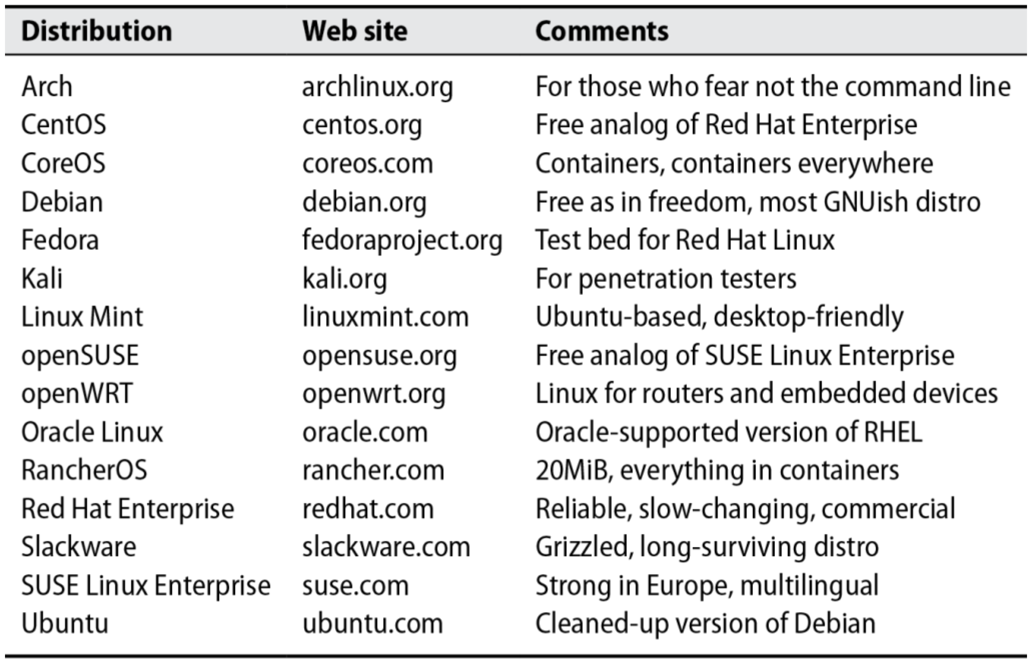
A Linux distribution comprises the Linux kernel, which is the core of the operating system, and packages that make up all the commands you can run on the system. All distributions share the same kernel lineage, but the format, type, and number of packages differ quite a bit. Distributions also vary in their focus, support, and popularity. There continue to be hundreds of independent Linux distributions, but our sense is that distributions derived from the Debian and Red Hat lineages will predominate in production environments in the years ahead.

By and large, the differences among Linux distributions are not cosmically significant. In fact, it is something of a mystery why so many different distributions exist, each claiming “easy installation” and “a massive software library” as its distinguishing features. It’s hard to avoid the conclusion that people just like to make new Linux distributions.

By adopting a distribution, you are making an investment in a particular vendor’s way of doing things. Instead of looking only at the features of the installed software, it’s wise to consider how your organization and that vendor are going to work with each other. Some important questions to ask are:

- Is this distribution going to be around in five years?
- Is this distribution going to stay on top of the latest security patches?
- Does this distribution have an active community and sufficient documentation?
- If I have problems, will the vendor talk to me, and how much will that cost?

Table 1.1 lists some of the most popular mainstream distributions.



Distribution	Web site	Comments
Arch	archlinux.org	For those who fear not the command line
CentOS	centos.org	Free analog of Red Hat Enterprise
CoreOS	coreos.com	Containers, containers everywhere
Debian	debian.org	Free as in freedom, most GNUish distro
Fedora	fedoraproject.org	Test bed for Red Hat Linux
Kali	kali.org	For penetration testers
Linux Mint	linuxmint.com	Ubuntu-based, desktop-friendly
openSUSE	opensuse.org	Free analog of SUSE Linux Enterprise
openWRT	openwrt.org	Linux for routers and embedded devices
Oracle Linux	oracle.com	Oracle-supported version of RHEL
RancherOS	rancher.com	20MiB, everything in containers
Red Hat Enterprise	redhat.com	Reliable, slow-changing, commercial
Slackware	slackware.com	Grizzled, long-surviving distro
SUSE Linux Enterprise	suse.com	Strong in Europe, multilingual
Ubuntu	ubuntu.com	Cleaned-up version of Debian

Table 1.1: Most popular general-purpose Linux distributions

The most viable distributions are not necessarily the most corporate. For example, we expect Debian Linux to remain viable for a long time despite the fact that Debian is not a company, doesn’t sell anything, and offers no enterprise-level support. Debian benefits from a committed group of contributors and from the enormous popularity of the Ubuntu distribution, which is based on it.

Although there are dozens of dialects of Unix, there are hundreds of different Linux distributions. Navigating between the available dialects can be challenging. Nearly all of the dialects can be categorized into one of four ancestor paths.

- *Debian*: This branch includes the very popular Ubuntu which itself has spawned dozens of subdialects.
- *Red Hat*: There are as many or more subdialects of Red Hat as Debian. The most popular subdialect is Fedora. Another popular descendant is Mandrake and another is CentOS.

- *SLS/Slackware*: This branch was first produced by a German company which led to SuSE Linux. Although there are dozens of spin-offs of SLS/Slackware, it is far less popular than either Debian or Red Hat.
- *Miscellany*: There are dozens of dialects that either led nowhere or have few successors.

Videos

Introducing Ubuntu: https://www.linkedin.com/learning/learning-ubuntu-desktop/introducing-ubuntu?u=2133849

What's Linux?: https://www.linkedin.com/learning/linux-tips-weekly/foundations-what-s-linux?u=2133849

Distros: https://www.linkedin.com/learning/linux-tips-weekly/foundations-distros?u=2133849