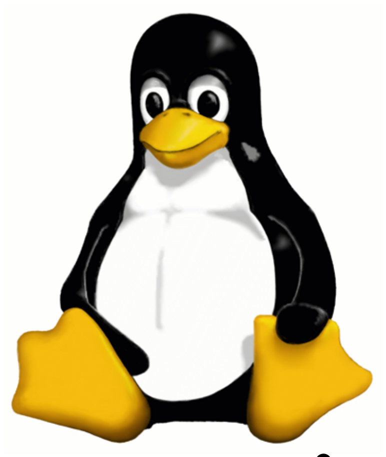

Intro to Linux

A Brief, Non-Technical Introduction to GNU/Linux

By Daniel Yim
For the SFASU Computer Science Club

Workshop Objectives

In this workshop we will focus on an introductory approach to Linux and other *nix systems.

By the end of this tutorial, you will have learned:

- The rich history of Unix, the Free Software Foundation and the GNU, the Linux kernel, and GNU/Linux
- The major Linux distributions
- The Linux file system organization
- Basic commands for command-line navigation
- Advanced command techniques

1. A Short History of GNU/Linux

MULTICS (1964) $\rightarrow$ Unix (1969-1970) $\rightarrow$ MINIX (1987) $\rightarrow$ GNU/Linux (1991)

Timeline of *nix operating systems and when they were first developed

In 1969 scientists at AT&T Bell Labs began developing an experimental operating system called **Unix** in the C programming language, which was a visionary move at the time. Due to Unix's use of a high-level machine-independent language, Unix became the first widely used multi-platform operating system.

As iterations of Unix releases became heavily commercialized, Richard Stallman, a free software advocate, and his company, the Free Software Foundation (FSF), began the **GNU project** in 1984, which was a project that sought to create a freely usable, distributable, and modifiable version of the Unix operating system. Although many useful components for the project were created (such as the GNU C library, the Emacs editor, Bourne Again Shell), the *core* of an operating system—the kernel—proved to be the most troublesome to develop. Then along came an astute Finnish student named Linus Torvalds, who provided the final piece of the free software puzzle.[1]

When a minimalistic Unix-like operating system, **MINIX**, was written by Andrew Tanenbaum in 1987 as a proof-of-concept for his textbook, Torvalds used ideas from MINIX and began developing a free Unix-like kernel, which he dubbed the "**Linux**" kernel after himself. Torvolds' creation combined with the efforts of the FSF to create a revolutionary open-source operating system called the **GNU/Linux** operating system, or more commonly (and mistakingly) known just as **Linux**.[2]

2. Linux Distributions

The Linux operating system has come a long way since its inception into the computing community. Many companies and/or interest groups took what Linux did to its predecessor and customized the operating system to suit their specific needs. These custom interpretations are known as **distributions**. Though every distribution is different in some way, all Linux distributions are unified by a single component: the Linux kernel. Some distributions may or may not include specific packages or GUIs, but they will *always* have the same underlying structure (bash, GNU's glibc, etc.).

Here is a partial list of popular Linux distributions: [3]

Red Hat	Fedora	Ubuntu	CentOS
Slackware	Debian	Mandriva	openSUSE
Gentoo	Knoppix	Yellow Dog	...and many others

For new users, many Linux veterans recommend **Ubuntu** (*http://ubuntu.com*), as it is the most user-friendly and the easiest for transitioning from a GUI-heavy Windows or Mac environment. If you are feeling particularly adventurous, you should try compiling a distribution from source or creating your own!

3. File System Organization

Let's take a look at the contents of a basic Linux root (/) directory:

```
[user@computer ~]$ ls
bin   dev  home  lost+found  misc  net  proc  sbin  srv  usr
boot  etc  lib   media       mnt   opt  root  sys   tmp  var
```

In this example, we see 10 directories (or subdirectories) in the root directory (/). Note that most of these directories are the same across all Linux distributions.

The neat thing about any Unix/Unix-like operating system is that the OS stores every entity it has control over in the form of a file. Even hardware devices such as USB ports or printers are stored under the `/dev` directory as **device nodes**. This Unix-exclusive feature promotes transparency between the OS and the user.

Here is a summary of what each directory holds[4]**:**

`/bin`

The bare minimum set of binary executable programs required for a user to use the system is stored here. We can expect crucial programs such as the filesystem navigation commands and the shell (`bash`) to reside here.

`/dev`

As stated before, everything in Linux is treated as a file, even hardware devices like USB ports, hard drives, and scanners. In order to access these devices, a special file called a device node has to be present. All device nodes are stored in the `/dev` directory.

`/etc`

This directory holds system configuration files.

`/home`

Linux is a multiuser operating system. Each user on the system is given an account and a unique directory for personal files. This directory is called the user's home directory.

`/lib`

System libraries that are required for basic operation are stored here. The C library, the dynamic loader, the ncurses library, and kernel modules are among the things stored here.

`/mnt`

This directory contains temporary mount points for working on hard disks or removable drives. Here you'll also find mount points for your CD-ROM and floppy drives.

`/proc`

This is a unique directory. It's not really part of the filesystem, but a virtual filesystem that provides access to kernel information. Various pieces of information that the kernel wants you to know are conveyed to you through files in the /proc directory. You can also send information to the kernel through some of these files. Try doing cat /proc/cpuinfo.

`/root`

The system administrator is known as `root` on the system. `root`'s home directory is kept in `/root` instead of `/home/root`.

`/tmp`

The temporary storage location. All files placed in here will automatically be deleted eventually.

`/usr`

This is the big directory on a Linux system. Everything else pretty much goes here, programs, documentation, the kernel source code, and the X Window system. This is the directory to which you will most likely be installing programs.

`/var`

System log files, cache data, and program lock files are stored here. This is the directory for frequently-changing data.

4. The Command Line Interface

Unlike the graphical user interfaces (**GUIs**–pronounced "gooey") that we're accustomed to, we will now introduce ourselves to the powerful command line interface (**CLI**). Although the GUI is enough to do most tasks, the command line offers more control and flexibility. This applies especially for Linux users, as familiarizing ourselves with the command line is very important for executing advanced tasks easily and efficiently.

In the world of Linux, GUIs may vary between distributions or user preference (the three most popular are **KDE**, **GNOME**, and **Xfce**), but for CLIs, all Linux distributions have the **the Bourne-Again Shell**, or simply `bash` (**B**ourne-**A**gain **Sh**ell). Given that information, we can assume that if we acquaint ourselves with `bash`, we'll be able to use almost any Linux CLI, regardless of distribution.

Home Folders

Before we get to commands, we must briefly discuss file structures in Linux. Unix (and Linux) is a multiuser system, meaning that many users can be logged on and use the system at the same time. Amongst directories that are shared across all users, Unix developers also allocated *home* directories (which in Linux is denoted as a tilde ~) for each user, and all home directories reside within the **root** (which is denoted as a single forward slash /) directory.

Pathing

We will now explore *pathing* in Linux systems. First, we know that on a Windows system, a path may look something like `C:\Windows\System32`, where `C:\` denotes a drive letter and backslashes (\ ) separate levels of folders. In Linux/Unix, however, we use forward slashes (/) to separate levels of directories; therefore, a path in Linux would look something like `/usr/lib/somefile`.

There are two types of pathing: **absolute** (e.g. `/usr/lib/somefile`) and **relative** (e.g. `usr/lib/somefile`). The only difference between the two is the preceding forward slash (/) which means to begin traversing directories from the root (/), whereas no preceding forward slash means to begin traversing from the current directory.

That may be hard to understand at first, so let's see an example of a hypothetical file structure, where indentations denote directory levels:

```
/
 bin
 usr
   John
     music
   Jane
     documents
     programs
```

If we are currently in John's music directory (`/usr/John/music`) and we want to refer to Jane's programs directory, we can denote that by an absolute path (`/usr/Jane/programs/` – start from the root, go to usr, go to Jane, go to programs) or a relative path (`../../Jane/programs` – start from John's music directory, go one directory up from the current directory (to `/usr/John/`), go another directory up from the current directory (to `/usr/`), go to Jane, go to programs).

Commands

In `bash`, a typical command follows this arrangement:

```
[user@computer ~]$ man   -k   grep
   |      |      |     |      |      |
username |      | command  |      |
   workstation |        option   |
             location        argument
```

We see that we have a command (required), an option after a dash (optional), and an argument (optional/required depending on the command), all of which are separated by spaces. This format applies for all Unix/Linux commands within a typical Linux shell.

`bash` has a long history as to how it came about with that name, but we know that Unix users did not enjoy typing much. What we see often with `bash` commands is that they're heavily abbreviated products of real, meaningful names. For instance, `pwd` stands for Print Working Directory, `man` stands for Manual, and `grep` stands for Global Regular Expression Print. In the next section, we will explore some of these commands.

5. Basic Bash Commands

Let's explore the first set of commands we'll learn:

File Copy, Move, and Remove (cp *target_file destination*, mv *target_file destination*, rm *target_file*)

```
[user@computer ~]$ cp somefile.ext somelocation
[user@computer ~]$ mv somefile.ext somelocation
[user@computer ~]$ rm somefile.ext
```

Here are some simple commands to handle files. For commands requiring a location, you can use a dot (.) to reference your home directory or a forward slash (/) to reference the root directory. For example: `cp code.cpp .` will move the file code.cpp from the current directory to your home directory.

Print Working Directory (pwd)
Prints the full path for the current directory.

```
[user@computer ~]$ pwd
/usr/user/home/
```

Directory Listing (ls)
Lists all the files within a given directory.

```
[user@computer ~]$ ls
Desktop  mail  mbox  public_html
```

If no argument is provided, it will list the current directory's contents, excluding hidden files (unless the `-a` option is used). Let's see what happens when we use the "show all" option `-a`:

```
[user@computer ~]$ ls -a
.               .config      .gconfd          .local     public_html
..              Desktop      .gnome2          mail       .qt
.bash_history   .dmrc        .gnome2_private  mbox       .redhat
```

```
.bash_logout    .emacs.d      .ICEauthority    .mcop      .viminfo
.bash_profile   .fonts.conf   .kde             .mozilla   .Xauthority
.bashrc         .gconf        .lesshst         .nx        .xcompmgrrc
```

The hidden files in the directory are denoted with a leading dot. Notice that in our `ls -a` output, we have two directories that we haven't seen before: the . directory (home folder) and the .. directory (one directory up).

Change Current Directory (cd *destination*)

Changes the current directory to the argument.

```
[user@computer ~]$ pwd
/usr/user/home/
[user@computer ~]$ cd /
[user@computer ~]$ pwd
/
[user@computer ~]$ cd ~/music
[user@computer ~]$ pwd
/usr/user/home/music
[user@computer ~]$ cd ../../../user2/
[user@computer ~]$ pwd
/usr/user2
```

Make/Remove Directory (mkdir *path*, rmdir *path*)

Makes/removes a directory in the path provided.

```
[user@computer ~]$ mkdir music
[user@computer ~]$ ls -a
.  ..  bin  dev  etc  home  mnt  music  proc  root  tmp  usr
[user@computer ~]$ rmdir music
[user@computer ~]$ ls -a
.  ..  bin  dev  etc  home  mnt  proc  root  tmp  usr
```

To make or remove multiple directories, you can use a space between each directory name.

```
[user@computer ~]$ mkdir music school_work pictures files
[user@computer ~]$ ls -a
.   bin  etc    home  music     proc  school_work  usr
..  dev  files  mnt   pictures  root  tmp
[user@computer ~]$ rmdir pictures school
[user@computer ~]$ ls -a
.  ..  bin  dev  etc  files  home  mnt  proc  root  school_work  tmp
   usr
```

If you want to delete all directories that you created (assuming that they're empty), you can try using a * wildcard.

```
[user@computer ~]$ rmdir *
[user@computer ~]$ ls -a
.  ..  bin  dev  etc  files  home  mnt  proc  root  school_work  tmp
   usr
[user@computer ~]$ ls -a
.  ..
```

UNIX Manual Pages (man *command*)

Shows the documentation for the specified command.

For example, let's say we're curious as to what the `which` command does. We'll try entering `man which` into `bash` and be presented with a full screen of information, which we can exit out by typing `q` then pressing Enter.

One thing amazing about Unix/Linux is that almost every program in the system is well-documented through the `man` pages. All `man` pages will describe the behavior of the command and what arguments and options it will accept.

Concatenate (cat *file*)
Prints the contents of a file.

```
[user@computer ~]$ cat file.txt
This is a test file with test data!
cat is a wonderful command.
```

The `cat` command is very useful for quickly checking the contents of a file without opening an editor, whether it may be a system information file (`/proc/cpuinfo`) or a plain text file.

6. Advanced Bash Techniques

The "commands" we will learn now are not entirely commands but are rather some new syntax which will introduce us to a new level of CLI usage. We explore new `bash` syntax and ways of using them.

Output Redirection Operators (*command1* | *command2*, *command* > *file*, *command* >> *file*)

The pipe (|) will redirect the resulting output of the first command as the input to the second command. The single chevron (>) will write the output of the command to a file, overwriting anything inside of it. On the other hand, the double chevron (>>) will append (add to the end) the output of the command to the file.

```
[user@computer ~]$ cat /proc/cpuinfo | less
<< full-screen of data >>

[user@computer ~]$ cat sampletext
This is a sample text file.
[user@computer ~]$ cat sampletext2
Another sample text file.
[user@computer ~]$ cat sampletext2 >> sampletext
[user@computer ~]$ cat sampletext
This is a sample text file.
Another sample text file.
[user@computer ~]$ cat sampletext2 > sampletext
[user@computer ~]$ cat sampletext
Another sample text file.
```

Global regular expression print (grep *string*)

Prints the line that matches the regular expression string provided.

```
[user@computer ~]$ cat /proc/partitions
major minor  #blocks  name

   8     0  214722560 sda
   8     1  214716726 sda1
   8    16   71567360 sdb
   8    17     104391 sdb1
   8    18   71457120 sdb2
   8    32   71567360 sdc
   8    33   16779861 sdc1
   8    34   54781650 sdc2

[user@computer ~]$ cat /proc/partitions | grep "sdc2"
   8    34   54781650 sdc2
```

The `grep` command is used to extract lines of interest from a large file. It is a very useful command that is almost always coupled with pipes. In our example above, instead of displaying a file with 10 lines of irrelevant information, we print only the lines that match the string we "grepped".

A common option is the case insensitivity option (`-i`). For example, the command `cat somefile | grep -i "elEPHanT"` will display the lines of the file `somefile` that contain any instance of the word "elephant" regardless of case.

Less (*command* | less)

Displays the output of the command in a scrollable window.

```
[user@computer ~]$ cat /proc/cpuinfo | less
```

Instead of displaying a large text file through a `cat` command in the console, piping the output to `less` is common. The `less` command converts the console into a simulated scrollable screen (which can be controlled by the up/down arrow or page up/down keys). Text files that are more than a screen's worth of lines can be displayed in a manageable manner this way.

Conclusion

Although what we covered is only intended to be a crash-course introduction to GNU/Linux, the information provided in this article should be sufficient to make a beginner become more or less of an "average" Linux user. The best way to learn this complicated yet powerful operating system is to simply play with its features, as that knowledge will grow well for the user.

Additional Resources

- **Ubuntu** *http://ubuntu.com*
  Ubuntu (pronounced ooh-BOON-tu) is a popular distribution that is relatively easy to install and is beginner friendly. As with most Linux distributions, Ubuntu is free to use and distribute, so I would suggest Ubuntu if you're curious about trying it on your system.

- **Slackbook** *http://slackbook.org/html/index.html*
  The "Slackbook" is an amazing guide put together by Linux veterans to assist new users of the Slackware distribution. The material presented in Slackbook is very practical, and most of the concepts presented are universal and not specific to just Slackware.

- **Google Code University** *http://code.google.com/edu/tools101/index.html*
  Google provides three great easy-to-follow tutorials on the material we covered as well as other topics of Linux. These guides are excellent for reinforcing what you've learned from this article.

References

1. Wheeler, David. "History of Unix, Linux, and Open Source / Free Software."
   < *http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/history.html* >.

2. "Linux." Wikipedia, The Free Encyclopedia.
   < *http://en.wikipedia.org/w/index.php?title=Linux&oldid=325431047* >.

3. "Linux distribution." Wikipedia, The Free Encyclopedia.
   < *http://en.wikipedia.org/w/index.php?title=Linux_distribution&oldid=325234422* >.

4. "Slackbook, Slackware Linux Essentials." Slackware Linux Inc.
   < *http://slackbook.org/html/index.html* >.

5. "Basic Linux Commands." Google Code University. Google Inc.
   < *http://code.google.com/edu/tools101/linux/basics.html* >.

6. "File System Layout." "Slackbook, Slackware Linux Essentials." Slackware Linux Inc.
   < *http://slackbook.org/html/system-configuration.html* >.

This document was last revised March 27, 2010.