LINUX

LINUX FOR BEGINNERS GUIDE

TO LEARN LINUX COMMAND LINE, LINUX OPERATING SYSTEM AND LINUX COMMANDS

JOSH THOMPSONS

Linux:

Linux For Beginners Guide to Learn Linux Command Line, Linux Operating System And Linux Commands

the trademark is without permission or backing by the trademark owner. All trademarks and brands within this book are for clarifying purposes only and are the owned by the owners themselves, not affiliated with this document.

Contents

Introduction

I want to thank you and congratulate you for purchasing the book, *"Linux: Linux For Beginners Guide to Learn Linux Command Line, Linux Operating System, And Linux Commands."*

This book contains proven steps and strategies on learning what Linux is and how to use it. Linux is, without a doubt, the best and most often used operating system that is open-source. It sits beneath all the other software that is on your computer, getting requests from applications and programs and relaying them on to the hardware.

Linux was created by Linus Torvalds in 1991. At the time, he was studying at the University of Helsinki and he created Linux as an open-source and free alternative to Minix, a Unix clone used in academia. Originally, he was going to call it Freax but the administrator of the server that Torvalds was using to distribute his code names the directory Linux, a mix up of Linus and Unix and that name has stuck ever since.

Most people use Linux, whether they are aware of the fact or not. How? Because many of the web pages you find on the internet today are run from Linux servers. These servers are secure which is why many companies opt to use Linux and there is also an excellent support system in the form of a large user and contributor community. If you own an Android device, a personal video recorder, a camera, a digital storage device, even a car, you are using Linux.

Unlike Mac OS and Windows, Linux comes in many different formats, known as distributions or distros. These are collections of software made into an

operating system that is based on the original Linux kernel – the fact that Linux is open-source means that anyone can take the source code and remold it for their own use.

Throughout this book, I will be providing you with plenty of hands-on examples and a few exercises for you to try your hand at. These will help compound what you learn. Also at the end of each of the chapters, I have included a quick quiz – you will find the answers at the end of the book

Thanks again for purchasing this book, I hope you enjoy it!

Chapter 1: Installing Virtual Machines

VirtualBox is a virtual machine that, first developed by Sun MicroSystems, is now under the ownership of Oracle. It simulates a separate computer and each virtual machine can have its own applications, operating system, and a whole host of other things. VirtualBox is ideal for testing out various operating systems, in this case, Linux on a Windows or Mac OS computer. By using Linux in this way, you don't need to make any permanent changes to your current system. We're going to look at how to install VirtualBox on Windows and Mac

Windows

- Go to the VirtualBox Download page and find the latest version; click on it
- On the next page, look for the file that ends .exe and download it – remember the location you saved it to on your computer
- Once the installer has been downloaded, double-click on the .exe file and follow the instructions on the screen to install it onto Windows – be aware that you may lose your network connection for a while during the installation because virtual network adaptors are being installed.
- Now you must reboot your computer and you should find VirtualBox in your apps. From here, you can run it and install any other operating system that you want to try.

Mac OS

- Go the VirtualBox download page and download the latest version of the app for the Mac

- Save the .dmg to a file location that you will easily remember – make sure you download the OS X hosts version
- Locate the file and install it using the executable file
- Reboot your computer and you can start using VirtualBox

Installing Linux Using an Image for VirtualBox

Windows

After you have followed the above steps to install VirtualBox to your computer, you need to download the disk image for Ubuntu Linux

- First, if you haven't already got a BitTorrent client installed on your computer, download one now – BitTorrent is a P2P application that allows downloads from other users, significantly easing the loading in the Ubuntu servers.
- Now head to the Ubuntu release website and download the latest release version – do NOT click on any links for Desktop CD. You will find a full list of links at the bottom of the page and make sure you click one with the .iso.torrent extension – download it to a location you will remember
- Now copy that to a bootable USB

A note of caution here – if you have got WinRAR installed, it will automatically associate itself with the file you downloaded and will ask you if you want to use WinRAR to extract the contents – do NOT use WinRAR and do NOT extract the .iso

IMPORTANT – Before you start the next step back up the contents of your hard drive somewhere safe! If you don't you will lose everything!

- Open VirtualBox from your Start menu and click New – this will open the New Virtual Machine Wizard

- Click on Next and give your virtual machine a name – stick with Ubuntu or Linux for ease
- If you have more than 1 GB of RAM on your computer, allocate one-quarter maximum to the virtual machine – if you have less, stick with what VirtualBox recommends. Click Next
- Click on Create New Hard Disk and then on Next
- Click on Next again and you will come to a screen where you set the type of hard drive. Choose Fixed-Size Storage
- If you are intending to add software or to install potentially large files in virtual Linux, add on some buffer; click Create
- Once the virtual hard drive has been created, you need to add the .iso image you downloaded. Click Settings>Storage>CD/DVD Device
- Where it says Empty, there is a folder icon so click on it
- Choose the .iso that you already downloaded and click on OK
- Now double-click on the virtual machine so it starts
- You will more than likely get a load of instructions and warnings about using guest operating systems – read them and then mark them so they don't come up again.
- Wait while Ubuntu loads
- Before you can install Linux, you must first change your BIOS settings on your computer – usually, when you start your computer, you hit F1, F2, F12, Escape or Delete. Restart your computer now and get into the BIOS settings and change the boot option to boot from USB first
- Now plug your USB stick in and reboot your computer again
- You will see a screen that is blank except for a few logos at the bottom
- Press any key and a new screen will appear – choose the language you want
- Now click on Install Ubuntu

- The installer will start and you may be asked to choose your language again
- Tick the option for installing closed source software
- Now you will be asked to connect to your Wi-Fi if you aren't connected already – you don't need to do this right now and it will make your installation take longer so ignore it
- There are three options here – choose the first one and then drag the slider to choose the hard drive sizes for Windows and for Ubuntu. Your hard drive will now be partitioned
- Answer the questions that appear on your screen as they appear – these are self-explanatory
- Now you just wait for Ubuntu to be installed – depending on the speed of your computer this can take up to one hour
- Reboot your computer and remove your USB stick – when you boot up again you will be in a working Ubuntu environment

If you changed the size of your partition for Windows, you will be asked to do a disk check – this is not necessary. Go back to Settings>Storage>CD/DVD and check that it has the Empty entry again – this will eliminate the need to use your USB every time you boot your computer.

Mac OS

When the VirtualBox installation has completed, you need to download the iso

- Go to the Ubuntu download page and click on the Mac iso image – choose your geographical location and then click on Begin Download. Make sure you save the file and not open or mount it
- Open VirtualBox and register it
- Now you can create a new virtual machine – click on New to open the wizard

- Click Next and type in a name for the virtual machine
- Choose Linux Ubuntu from the operating system menu and click on Next
- Set your base memory as 384 MB and click on Next
- Now you need to click to create a new hard disk which is just a disk that takes up space on your existing drive – make sure you have enough to do this! Accept all the default settings and click on Next
- The Create New Virtual Disk wizard will appear, click on Next
- Click on Dynamically Expanding Storage and click on Next
- Choose where you want your hard drive to be and how big – 2 TB is the maximum although 8 GB is more than sufficient
- Click on Next and then on Finish
- The framework is ready so make sure you have copied the iso to a bootable USB or DVD disk

VirtualBox will now show you that you have a virtual machine with the name of Ubuntu Linux.

- Insert your DVD or USB
- Click on CD/DVD and then tick the box for Mount CD/DVD drive
- Click on ISO Image File
- Click the folder beside No Media and then drag your iso file into it
- Click on Select and then on OK
- Click on Start and black screen will appear – this is the new session and everything you do now will be done on the virtual machine – if you want to change that back to your Mac just press the Command key on the left side of the space bar
- Click OK
- When the Ubuntu screen appears, double click on Install and answer the questions that appear on the screen
- When finished click on Forward

- Now accept the default values for where Ubuntu is going and click on Forward
- Input your personal information, including a new password and a name for your machine
- Choose whether to log in automatically or use a password and click on Forward
- Click Install and wait
- When the installation has finished, you will either be asked to use your password to go in or you will automatically be at the Linux desktop, depending on what option you chose
- Ubuntu will now check for updates so click on Install Updates and wait
- Reboot your Mac is needed and you are ready to go

How to Install CentOS from Scratch

CentOS is a stable Linux distribution and here's how to install it:

- Go to the download page for CentOS and download the ISO
- Now make a bootable drive, transferring the ISO to a formatted USB stick. To do this, plug your USB in and open a command prompt window. Type the following in - # dd if=/iso/CentOS-7-x86_64-DVD-1602-99.iso of=/dev/sdb – ensure that you have a minimum 4.3 GB space on the USB stick
- On the desktop, click on Install to Hard Drive
- Choose your keyboard type and preferred language – make sure you pick the right keyboard or some of your keys will be scrambled up
- The default for the installer is to choose Automatic Partitioning of your hard drive – click the icon for Installation Destination and change that to Custom Partitioning
- Choose the hard drive to install CentOS to and then click on Other

Storage Options

- Click on I Will Configure Partitioning and then on Done
- Choose Standard Partition
- To create a Swap Space on one of your partitions, select the File System for the swap space and name it as swap and then click on Reformat
- Now you need to create your mount point – this is where the root partition is going to be installed. Set your mount point and then set Label and Desired Capacity to whatever you require them.
- Click on ext4 to set the file system and then on Reformat
- Click on Done and accept the changes you have made
- Click the clock and set your time zone; click Done
- Click on Begin Installation; as it goes through, follow all the onscreen instructions, setting up your user account and the root password. To do this, click Root Password and type in the password twice; click on Done
- To create your user account, input the details and, if this is going to be an administrator account, ensure that you tick the options for Make This User Administrator and on Require a Password
- Once the installation is complete, you will see a message telling you it was successful; click on Quit
- Log out and then log in to your new CentOS installation
- Accept the EULA and you are ready to go

Connecting Your Linux System over the Network

If you are looking to connect your Linux system over your network, you must use something like SSH. This is an acronym for Secure Shell and it is one of the most well-known of the network protocols. The purpose of SSH is to let you connect securely to remote machines on your network. To connect your Linux system over your network:

- **Windows –** use PuTTY
- **Mac and Linux –** use ssh on the command line

Windows

- Open your web browser and download PuTTY
- Open PuTTY and type in the IP address or Hostname into the correct box.
- If you had no port number provided, leave it as the default of 22
- Click on Data and then on the Auto-Login Username – input your username
- To save a session, click the Saved Sessions box, type in a name and click on Save in the future you will be able to double click the saved session to connect
- Click on Open and a connection is made. When you connect to a server for the first time, PuTTY will ask for permission to cache the host key for the server – click on Yes

Linux and Mac

Both Linux and Mac already have the SSH client built in as a command line program. To get to it run the terminal:

- **Mac –** Applications folder
- **Linux –** Open Dashboard and search for Terminal

When the terminal has been started simply use the ssh command to get your connection. Do be aware that commands are always case-sensitive so only use lowercase. Type in ssh followed by the username on the Linux server

When you connect for the first time to a server, you will need to verify the host key. Type Yes to continue connecting and the press Enter. When the connection is established, input your password.

To get out of the connection, type in exit and then log out

Quick Quiz 1

1. What is VirtualBox?
2. What is CentOS?
3. What does ssh stand for?
4. How do you access the BIOS settings on your computer?

Chapter 2: The Linux Directory Structure

If you are coming to Linux from a Windows system, the directory structure will seem somewhat strange. There are no longer any drive letters, these are replaced with a/ and some strange sounding directory names, most of which are just three letters. The FHS, or File Hierarchy System, is what defines the file system structure on Linux but there are also a few directories that have not yet got a definition under this standard:

/ – The Root Directory

On Linux, everything is found under the directory / which is also called the root directory. This is like the C:/ on Windows but without any drive letters.

/bin – Essential User Binaries

This directory is home to the binaries or programs that are essential to the system being mounted in single-user mode. Things like Firefox would be stored in the directory called /usr/bin, while the utilities and system programs, like bash shell, are found in /bin. /

/boot – Static Boot Files

This is where the files needed to boot up your system are located, such as the GRUB boot loader and the kernel files. The configuration files, however, are in /etc.

/cdrom –Mount Point for CD-ROMs

This directory doesn't come under the FHS standard but is still to be found in most operating systems. It is a temporary location for when you insert a CD-ROM into your computer

/dev – Device Files

Devices are exposed as files on Linus and this directory has several special files that are representative of devices. These are not files in the way you know them but they look like they are. For example, if you see /dev/sda, it is representing the initial SATA drive in your system. You will also find pseudo devices here, virtual devices that don't correspond to any hardware.

/etc – Configuration Files

This is where the configuration files are found and these can be edited in any text editor. However, only the configurations of the system are found here; if you have user-specific ones, they will be found in the user's home directory

/home – Home Folders

This is where you will find a home folder for each of the users. This contains the data files and configuration files specific to the user and each user can only write to their own home folder. If you want to modify others, you need elevated permissions, i.e. to become the root user

/lib – Essential Shared Libraries

This contains all the libraries that the binaries located in /bin and /sbin need.

/lost+found – Recovered Files

Every file system has a lost and found directory; if your system were to crash, at the next bootup, a file system check is performed. If corrupted files are found, they are put into the directory, giving you the chance to recover as much as you can.

/media – Removable Media

Here you will find subdirectories and this is where removable media that is connected to the system is mounted. For example, a BD, DVD or USB. When

each is mounted, a new drive will appear, allowing you to access the media contents.

/mnt – Temporary Mount Points

This is where administrators mount temp file systems while they are using them

/opt – Optional Packages

Here you will find subdirectories for software packages that are optional. It is generally used by software that doesn't follow the rules of the standard FSH

/proc – Kernel & Process Files

This is like the /dev directory as it contains no files that are standard; instead, it has special files that are representative of process and system information

/root – Root Home Directory

This is the root user's home directory. Note it is not located in /home/root, only at /root, distinguishing itself from /, which is the root directory for the system

/run – Application State Files

This is quite new and it provides a place for applications to store their transient files, like process IDs and sockets. These are not stored in /tmp because they may be deleted from there

/sbin – System Administration Binaries

Similar to /bin and contains binaries that are used for system administration by the root user

/selinux – SELinux Virtual File System

You will see this is your particular Linux distro used SELinux as security. The directory will contain the files needed by SELinux. If you are using Ubuntu and

you see this directory, it is a bug because Ubuntu does not use SELinux

/srv – Service Data

This is where you will find the "data for services provided by the system". If, for example, you were using the Apache HTTP server for your website, the files for the website would be stored in this directory

/tmp – Temporary Files

This is where temporary application files are stored and are deleted when the system is rebooted

/usr – User Binaries & Read-Only Data

Here you will find files and applications that are used by users and not by the system. For example, you would find applications that are not essential here, along with binaries that are not essential.

/var – Variable Data Files

Where /usr is only readable, /var is writable. Her you will find log files and anything else that is normally written to /usr

Different components of the operating system.

The Linux system is composed of three main code bodies:

1. **The Kernel –** this is in charge of the maintenance of all the abstractions of the Linux operating system, as well as processes and virtual memory. This is the center point for the operating system, the spine if you like, and is responsible for the functionality that is required to run any process, as well as providing system services that give protected access to your hardware resources. Anything that is required for the operating system to run is implemented by the kernel
2. **System Libraries –** these are a set of functions that allow interrelation

between applications and the kernel. These also apply a good deal of the functionality of the operating system that doesn't need the privileges or rights of the kernel code

3. **System Utilities –** these are the programs that execute specific managing tasks, those that are specialized. Some utilities may be invoked only once just to initialize features of your system and configure them. Others will run constantly, carrying out tasks like responding to network connections, both incoming and outgoing, accepting requests for logging on or updating files and log records.

Quick Quiz 2

Questions and Answers

1. Which directory contains a lot of the files that the programs use to run. This directory is needed to control the operating system.
    - /bin
    - /tmp
    - /proc
    - /lib
2. Which directory is used to store Kernel information needed to start the OS on startup?
    - /dev
    - /proc
    - /etc
    - /boot
3. What does the / directory signify?
    - Root directory
    - An empty directory
    - A new directory
4. What directory contains all the devices and their references in the file system?
    - /mnt
    - /lib.
    - /dev
    - /home
5. Where are the configuration files for the server's programs as well as the servers boot processes stored?
    - /lib
    - /dev
    - /proc

- /etc

6. Where is user data stored when a user is created on the server?
    - /root
    - **/home**
    - **/lib**
    - **/usr**

Chapter 3: All About the Shell

Our computers are programmed so that they understand a specific language. That language is called binary and it is comprised of zeros and ones. When computing first began, instructions would be provided in this language and that caused difficulty because it isn't a language that we can all read or write. To get around this, we have the shell and there is one in every operating system. The shell is what takes human commands and interprets them in a way that the kernel is able to read them and process them.

What is the Shell?

The shell is an environment or a user program for user interaction. Basically, it is an interpreter that translates the command language and executes the commands that are read from input devices like keyboards, or from a file. The shell is started when you open a terminal or when you log into your system. It is not a part of the Linux system kernel but it does make use of the kernel for creating files, executing programs, and lots of other activities. There are a number of shells that Linux users can make use of, including:

- **BASH -** free, open-source and the most common one in Linux
- **CSH –** with usage and syntax similar to C program language
- **KSH –** base of the standard specifications for the POSIX shell

Each shell does exactly the same but each will understand different syntax and will provide different functions.

The Shell Prompt

There are several ways to get shell access:

- **Terminal –** Linux has a GUI login system and, once you have logged in you can run one of several commands in the terminal to get access to shell – XTerm, Gterm or Kterm
- **SSH –** as soon as you log into a remote workstation or server, you will be given a shell prompt
- **Console –** Some Linux systems have a login system that is text-based and this will give you a shell prompt when you are logged in

To find out the name of your current shell or to find out which shells are available on your system, type in this command at the terminal prompt:

cat /etc/shells

If there is more than one shell listed, there is more than one that is supported by your particular platform.

Command Line Interface (CLI)

The shell gives you an interface with Linux where you can input commands with your keyboard and this is known as the CLI or command line interface. To find the type of the current shell in your system, input the following command:

echo $SHELL

ps $$

ps -p $$

Basic Command Line Editing

There are several key combinations that you can use to edit commands and to recall them:

- CTRL + L: Clear screen.
- CTRL + W: Delete the word that starts at the cursor.
- CTRL + U: Clear line of all words

- Up/Down arrow keys: Recall commands
- Tab: use to autocomplete the name of files, directories, commands, etc
- CTRL + R: Search commands you have used previously
- CTRL + C: Cancel any commands that are running
- CTRL + T: swap around the two characters that immediately precede the cursor
- ESC + T: Swap around the last two words that immediately precede the cursor
- CTRL + H: Delete the letter that starts at the cursor.

Executing A Command

Simply type your command in and press the enter key. For example, if you wanted to know the date, simply type in date and hit enter.

Command and File Completion

Bash Shell is able to automatically complete the names of commands and files when it can or when you tell it to. For example, type in sle and then press the tab key – the shell will complete the command automatically.

Getting Help in Linux

Most of the Linux commands have their own documentation with them and you can use the info or man command to view that documentation. For example, use the date command to open the manpage:

man date

You can also use the ls command to read information documentation:

info ls

Many of the commands will accept -h or –help as a command line option. For example, you can use the following to show the date command help options:

date --help

Basically, you can use any of the following to find out information about a Linux command:

- man commandName
- info commandName
- commandName -h
- commandName --help

Super User

Every Linux system has a root account, which is for the administrator. Many tasks require you to log on as administrator or execute commands using root privileges. If you do need to do this, be sure that you understand what commands you are running and the consequences each has. One careless command run under root privileges can have far-reaching effects, even going so far as to render your operating system useless and unusable.

Logging in as root

In many ways, the root account is the same as any user account in that you need a username and password to get into it. If you know the password for the root account you can use it to get into the root account via the command line. A special command that allows you to switch from normal to root user and the command is su – it stands for switch user or super user. Just type at the command line:

su

Input the password and, if it is correct, you will be switched over to the root user and will have system privileges to run commands. Always make sure you are aware that you are logged in as a root user – running some commands as root when you think you are logged in as a normal user, might cause damage to

your system. Get into the habit of checking the command prompt – if you see a dollar sign, you are in as a normal user, a # and you are logged in as root. When you are done with the admin tasks, simply type in exit or logout and you will go back to your normal account.

Running a Command as Root with No Password

Sometimes it is better not to log in as root but still run commands as the root and we do this by using another command, sudo. This means superuser do. If you put this in front of any command, you will be asked for your own password and then your details will be checked against a sudoers file. If you are in there, you can run commands with root privileges.

When you use sudo, it is more difficult to forget that you are acting as root because you won't be logged into the root account – this means never having to forget to log yourself out again. Also, if you have to type sudo before you type every command, especially ones that have the potential to do some serious damage, you are unlikely to run these commands without thinking.

Quick Quiz 3

1. Which of the following commands allows definition and assignment of environment variables under bash
    - Env
    - Export
    - environ
    - setenviron

2. Which variable contains current shell process id
    - $*
    - $?
    - $$
    - $!
3. Hidden files are
    - Those whose ‘read’ bit is set to ‘h’
    - Permitted for (can be accessed) only superusers
    - Files that begin with a ‘.’
    - Files that cannot be opened by ordinary user for writing
4. What is shell?
    - Command Interpreter
    - Interface between Kernel and Hardware
    - Interface between user and applications
    - Command Compiler

Chapter 4: Basic Linux Commands

This chapter is dedicated to some of the more essential but basic Linux commands that you need to know so fire up Linux and play along. There will be exercises to test your knowledge along the way, although I won't be providing answers to all of them because you should be able to work it out from the section you just read:

Listing Directories and Files

ls

When your login, you will always be in your home directory. This will have the same name as you have for your username and it is where all your personal files and subdirectories will be saved. To find out the contents of your home directory, type in:

% ls

You may not see any files in the home directory; if there aren't any, you will be returned to the prompt. Be aware that, using the ls command, you will only see the contents whose name does not start with a dot. The files that start with the dot are hidden files and will normally have some important configuration information in them. The reason they are hidden is because you should not be touching them.

To see all the files, including those with the dot, in your home directory, type in

% ls -a

You will now see all files including those that are hidden.

ls is one of those commands that is able to take options, and the above one, -a,

is just one of those options. These will change how the command works,

Making a Directory

mkdir

To make a subdirectory of the home directory, to hold the files you create, (for the purposes of this, we will call it linuxstuff, type in this in your current directory:

% mkdir linuxstuff

To see that directory, type in

% ls

Changing to Another Directory

cd

cd means change directory from the current one to directory so, to change to the directory you just created, you would type in:

% cd linuxstuff

To see the content, of which there shouldn't be any right now, type ls

Exercise

Go into the linuxstuff directory and then make another one called backups

. and .. Directories

Staying in the linuxstuff directory, type this in

% ls -a

You will see, and this is in all directories, two directories that are called . and ..

In Linux, a single dot (.) signifies the current directory so if you were to type in

(making sure to leave a space between cd and the single dot)

% cd .

you would stay exactly where you are in the linuxstuff directory

While this might not seem to have much use at first look, you will soon find that by typing a dot as the current directory name will save you quite a bit of typing

.. signifies the parent directory, which is the parent of the directory you are already in so if you were to type

% cd ..

you would go back one directory, in this case, to your home directory.

Note – if you get lost in your file system, simply type cd at the prompt and you will be returned straight to your home directory

Pathnames

pwd

pwd stands for print working directory and using a pathname lets you work out exactly where you are in the file system. For example, if you wanted to know the absolute pathname that goes with your home directory, you would type in cd, so you go bac to the home directory, and then type in

% pwd

You should see something like this as the pathname

/home/its/ug1/ee51vn

And this means that the home directory is in a subdirectory called ug1, which is a group directory and this is located in the subdirectory called its, which is located in the home subdirectory, in the top level of the root directory named /

Exercise

Explore your file system with the commands, cd, pwd and ls. Don't forget, typing cd will take you back to the home directory

Understanding Pathnames

Go back to your home directory if you aren't already there and type in

% ls linuxstuff

This will list the contents of the home directory. Now type in

% ls backups

You will see a message that says something like this:

backups: No such file or directory

Why? You created a directory with that name earlier but you didn't create it in the working directory. So, to get to backups directory, you either must use cd and specify the directory or you must use the pathname

% ls linuxstuff/backups

~ (your home directory name)

We can also use the tilde character (~) to refer to the home directory and to specify a path that starts at the home directory. So, if you typed in

% ls ~/linuxstuff

You would see a list of what is in the linuxstuff directory, irrelevant of where you currently are in the file system.

Exercise

Look at the following commands and work out what would be listed if you typed them:

% ls ~

% ls ~/..

Section Summary

Command	Meaning
ls	lists the files and the directories
ls -a	lists all directories and files including those hidden
mkdir	makes a new directory
cd directory	change to the directory named
cd	change back to the home directory
cd ~	change back to the home directory
cd ..	change to the parent directory
pwd	shows the pathname for the current directory

Copying Files

cp

If you wanted to copy file1 in the working directory and name it file2, you would type in

cp file1 file2

First, go to this website and copy the text into a file. Name it science.txt and save it to your linuxstuff directory

So, now we are going to copy a file that is to be found in an open access part of the file system to the linuxstuff directory. First, you would get back to your linuxstuff directory by typing

% cd ~/linuxstuff

Then you would type the following at the prompt

% cp /vol/examples/tutorial/science.txt .

Note – do not forget to add the dot at the end

The command is saying that we are going to copy the file called sceience.txt to linuxstuff but we will keep the name the same

The above command means copy the file science.txt to the current directory, keeping the name the same.

For the purposes of the next example, you must create a file named science.txt in your linuxstuff directory

Moving Files

mv

mv file1 file2 will move or rename file1 to file2

When you use the mv command, you will move the file and not copy it, ensuring that you still have just one file and not two. We can also use it to give a file a new name and we do this by moving it to the same directory it is already in but with a different name.

Go back to your linuxstuff directory and type in the following

% mv science.bak backups/.

Now type in ls and the ls backups and see what has happened

Removing a File or Directory

rm

rmdir

To delete a file, or remove it, we use the rm command. Let's make a copy of

science.txt and then we will delete it

From your linuxstuff directory, type in

% cp science.txt tempfile.txt
% ls
% rm tempfile.txt
% ls

If you want to remove an entire directory, first make sure there are no files in it and then use the rmdir command. Have a go at removing the directory called Backups – Linux won't allow it because it has something in it

Exercise

Use mkdir to create a new directory named tempstuff and then use the rmdir command to remove it

Displaying File Contents on the Screen

clear

Before we move on, lets clear our terminal window of all the commands already typed in so that we can better understand what the output of the next commands are. To do this, type

% clear

All the text will be removed and you will be left with the prompt. So, let's move on to the next command

cat

cat is used to concatenate and display a file's content on your screen. Type in

% cat science.txt

You will see that the file is bigger than the window size so it will scroll,

making the contents hard to read

less

This command will write the file contents to the screen one page at a time so type in

% less science.txt

Press on your space bar if you need to see the next page and, if you have read enough, type in q.

Note – if you have long files, use the command less rather than the command cat.

head

This command will write the first ten lines of the specified file to your screen. Clear your screen and the type in:

% head science.txt

Now type

% head -5 science.txt

Look at what you go and decide what adding -5 did to the command

tail

As opposed to the head command, the tail command will write the last ten lines of the specified file to the screen. Clear your screen and type in:

% tail science.txt

Looking Through a File's Contents

Using the less command, you can search for a keyword pattern in a text file. For example, if you wanted to find all the instances of science in the science.txt

file, you would type in

% less science.txt

And then, staying in less, you type a forward slash and the work you want to search:

/science

All the instances of the word are highlighted; to find the next instance, type in

grep

grep is one the standard utilities on Linux and it is used to search for specific patterns or words. Clear your screen and type in

% grep science science.txt

Now you can see that the command grep prints each of the lines that have the word science in it

Or has it?

Now type in

% grep Science science.txt

grep is case sensitive and will distinguish between science and Science. If you want to ignore this case sensitivity, use -i. For example, type in

% grep -i science science.txt

If you want to search for a specific pattern or phrase, it must be inside single quote marks. To search for spinning top, you would type in

% grep -i 'spinning top' science.txt

Other options with the grep command are:

-v will display the lines that don’t match the specified text

-n will precede each of the matching lines with the correct line number
-c will only print out the total number of the matched lines

Have a go at these and see what the results are. You can use more than one of these in one command so, to show the number of lines that do not include Science or science, you would type in

% grep -ivc science science.txt

wc

This is a neat utility and is short for word count. If you wanted to do a total word count on the science.txt file, you would type

% wc -w science.txt

If you want to know how many lines are in the file, type:

% wc -l science.txt

Section Summary

Command	**Meaning**
cp file1 file2	copies file 1 and names it file2
mv file1 file2	moves or renames file1 to file2
rm file	removes a file
rmdir	removes a directory
cat file	displays a file
less file	shows one page of a file at a time
head file	displays just the first 10 lines of a file (or however many specified)
tail file	displays the last 10 lines) or however many specified) of a

file

grep “keyword” file search for a specific keyword in a file

wc “keyword” file counts how many characters or words are in a file

Redirection

Most of the processes that are initiated by Linux commands will write to the terminal screen, which is the standard output. Many of them also take their input from the keyboard. As well as that, there are also those that write error messages to the terminal screen. Already, we have used the cat command to write a file’s contents to the terminal so now type the following, without specifying any file

% cat

Type a few words in using the keyboard, anything will do, and then press return

Hold down CTRL and press the d key – this will finish the input

When you run the cat command without a file, it will read the keyboard input and, when it receives the end of the file, the d, it will copy it to your terminal

In Linux, we are able to redirect input and output.

Redirecting Output

The . symbol is used to redirect command output. For example, if we wanted to create a file with a name of list1, that had a list of fruits in it, we would type:

% cat > list1

Then you type the names of a few fruits and, after each one, press return. For example

apple

pear

banana

then press ctrl+d

The cat command will read what was input from the keyboard and > will redirect it to the output, the screen, in a file named as list1. If you wanted to read what the file had in it, you would type

% cat list1

Exercise

Now, using the same method, create a file named list2, with these fruits in it – plum, orange, grapefruit, mango. Now read the file contents

Appending to Files

>> will append the standard output to a file so, if we wanted to add some more items to list1, we would type

% cat >> list1

And then the names of more fruits

grape

peach

orange

Then press CTRL+d to stop

To read the file contents, type

% cat list1

You should, by now, have two files, one containing six fruits and one

containing four fruits. Now we will join the two lists using the cat command into one file named biglist. Type in

% cat list1 list2 > biglist

This will read the contents of both lists, in turn, and then output the text from each into a new file called biglist

To read the contents of biglist, type in

% cat biglist

Redirecting Input

To redirect command input we use the < symbol. This will sort a list in numerical or alphabetical order. Type in

% sort

Now type some animal names in and press return after each of them:

ape

cat

dog

bird

then press CTRL+d to stop

The output would be

ape

bird

cat

dog

When you use < you can redirect input from a file instead of from the keyboard. For example, if you wanted a list of fruits sorted, you would type

% sort < biglist

The list will be sorted and output on the screen

If you wanted the sorted list to be output to a file, you would type

% sort < biglist > slist

The cat command is used for reading the contents of slist

Pipes

If you want to know who is on the same system as you, you would type in

% who

One way to get a list of names that has been sorted would be to type

% who > names.txt
% sort < names.txt

This is a rather slow method and you would need to remember that the temporary names file has to be removed when you are done. Really, what you are looking to do is connect

the output from the who command straight to the input of the command called sort. This is what pipes are for and the symbol for the pipe is a vertical bar (|). For example, if you typed in

% who | sort

You would get the same result but it would be much quicker

If you wanted to find out how many other users have logged in, type in

% who | wc -l

Exercise

Use pipes to show all of the lines in list1 and list2 that have the letter p in them and then sort the results

Answer

As this is a little more complex, I have opted to show you the answer this time:

% cat list1 list2 | grep p | sort

Section Summary

Command	**Meaning**
command > file	will redirect the standard output to a specified file
command >> file	will append the standard output to a specified file
command < file	will redirect the standard input from a specified file
command1 | command2	will pipe command1 output to command2 input
cat file1 file 2 > file0	will concatenate or join files 1 and 2 to file0
sort	will sort the data
who	will show you who is logged on to the system with you

Wildcards

* is a wildcard character and it will match with none or more characters in a directory or file name. For example, go to your linuxstuff directory and type in

% ls list*

This shows you all of the files that are in the current directory, beginning with list…

Now type in

% ls *list

This shows all the files that end with …list in the current directory

? is another wildcard character and it is used to match one character only. So, for example, if you were to type ?ouse, it would match with files like mouse or house, but it wouldn't match with grouse. Type in

% ls ?list

And see what happens

Filename Conventions

It is worth noting that directories are special file types so the naming conventions for files will also apply to a directory. When you name a file, you cannot use special characters, such as *, /, % and &. You also cannot use spaces so, when you name a file use numbers and letters, along with the underscore and the dot.

Good names	**Bad names**
project.txt	project
my_big_program.c	my big program c
bob_billy.doc	bob and billy.doc

File names begin with lowercase letters and end with a dot and a file extension that indicates the file contents. For example, if you have files that have C code n them, they may have the .c ending, such as prog1.c.

To list all the files that have C code I the home directory, all you need to type at the command prompt is ls*c. from within the home directory

Help

There are plenty of online manuals providing information about commands. The pages will tell you what a command can do, the options that it can take and how each of those options will modify the command. If you wanted to read the page for a specific command, you would type in man. For example, if you wanted to know more about the wc command, you would type in

% man wc

Or you could type

% whatis wc

This one would provide a short description of the command but wouldn't give you any other information about options, or anything else.

Apropos

When you do not know the name of the command exactly, you would type in

% apropos keyword

This will provide you all the commands with the word keyword in the page header in the help manual. Try typing:

% apropos copy

Section Summary

Command	**Meaning**
*	matches any amount of characters
?	matches just one character
man command	will read the page in the online manual for a specific command
whatis command	gives a short description of a specified command

apropos keyword	will match a command with a keyword in the man page

Quick Quiz 4

1. What command, followed by the directory name is used to access that specific directory?
    - Cp
    - Cd
    - Access
    - Acs
    - Cdr
2. What command is used to clear up the command prompt window?
    - Clr
    - Clrwin
    - Cd
    - Clear
    - Clearit
3. How would you show a list of files and directories that are inside the current directory?
    - List
    - Listfiles
    - Ls
    - Lst
    - Listout
4. What command shows the directory you are in?

- Pwd
- Dir
- Directory
- Showdir
- Dirpwd

Chapter 5: Teach Yourself Fish

The code used in this chapter is courtesy of https://fishshell.com

Fish is a command line shell, similar to Bash, that is incredibly user-friendly. It supports all sorts of very powerful features, such as syntax highlighting, tab completions, and autosuggestions, with no configuration needed and very little to learn. If you are looking for a command line that is more productive, a good deal more fun and definitely more useful, fish could be just the thing for you.

Learning Fish

I am going to assume that, by now, you have a basic understanding of shells and Linux commands. Download fish shell and let's get started. When you open fish, you should see a message the says:

Welcome to fish, the friendly interactive shell

Type help for instructions on how to use fish

you@hostname ~>

Fish already has a default prompt in it that will show your details – username, hostname and the current working directory. Later on, I will show you how to change the prompt but, for now, we will assume it is >

Running Commands in Fish

Running commands in fish is the same as in any other shell; you type the command and any arguments that go with it, using spaces as separators. Type in:

> echo hello world

And the output will be

hello world

To add a literal space in an argument, use a backslash or use single quotes or double quotes:

> mkdir My\ Files

> cp ~/Some\ File 'My Files'

> ls "My Files"

Some File

Help

Fish contains some very good man and help pages. Simply run help and it will open in a web browser, or if you run man, the help file will open in a man page. You can also find help on specific commands. For example, if you want man set to see help in the terminal:

> man set

set - handle shell variables

Synopsis...

Syntax Highlighting

You may already have noticed that fish highlights the syntax as you are typing it. If you see something highlighted in red, it is an invalid command:

> /bin/mkd

This may be because the command doesn't exist or because it is referencing a file that cannot be executed. When the command is valid, it will change color:

> /bin/mkdir

Valid file paths are underlined as they are typed

> cat ~/somefi

This will tell you that there is a file with the name beginning somefi , which is useful to you as you are typing.

Wildcards

The wildcard * is supported in fish. If you wanted to see a list of all the JPEG files, you would type

> ls *.jpg

leno.jpg

moona.jpg

santa margarita.jpg

You can use several wildcards:

> ls l*.p*

leno.png

lesson.pdf

The recursive wildcard is one of the most powerful as it will recursively search directories:

> ls /var/**.log

/var/log/system.log

/var/run/sntp.log

If you find it is taking too long to traverse the directory, simply use CTRL+C to get out of it

Pipes and Redirections

Fish supports the use of the vertical bar (|) as the pipe command and you can use it for piping between commands:

> echo hello world | wc

1 2 12

We can redirect stdout and stdin using the < and the >. Unlike the other shells, we use the caret (^) to redirect stderr:

> grep fish < /etc/shells > ~/output.txt ^ ~/errors.txt

Autosuggestions

As you type, fish will automatically suggest commands and this suggestion will be shown to the right of your cursor, highlighted in gray. For example:

> /bin/hostname

Fish also knows about options and paths:

> grep --ignore-case

And it knows about history. Type in a command and, when you want to use it again, simply start typing it in and the full command will appear:

> r<\@args{ync} \ ssh .
myname@somelonghost.com:/a/long/path/doo/dee/doo/dee/doo}

If you want to accept the suggestion press the right arrow key or CTRL+F. If you just want one word of the suggestion, hit Alt + the right arrow key. If the suggestion is not what you are looking for, simply ignore it.

To accept the autosuggestion, hit → or CTRL+F. To accept a single word of the autosuggestion, Alt → (right arrow). If the autosuggestion is not what you want, just ignore it.

Tab Completions

There are a lot of tab completions in fish that work without any configuration whatsoever. Simply start typing a command, path or argument and press the tab key to complete it:

> /pri @key{Tab} → /private/

If there are two or more suggestions, they will be listed:

> ~/stuff/s @key{Tab}

~/stuff/script.sh (Executable, 4.8kB) ~/stuff/sources/ (Directory)

Press the tab key to go through each suggestion until you get to the one you want.

Variables

As with other shells, the dollar sign is used to perform variable substitution:

> echo My home directory is $HOME

My home directory is /home/tutorial

You can also use double quotes but never single quotes

> echo "My current directory is $PWD"

My current directory is /home/tutorial

> echo 'My current directory is $PWD'

My current directory is $PWD

However, there is no syntax dedicated to setting a variable. Instead fish uses an ordinary command called set – this will take the variable name and then the value of the variable

> set name 'Master Noodle'

> echo $name

Master Noodle

Note the quotes. If they are not used, Master and Noodle would be classed as two separate arguments and $name would have been turned into two element lists. Also in fish, variables are not split after they have been substituted:

> mkdir $name

> ls

Master Noodle

In a Bash shell, you would have got two directories, Master and Noodle. In fish, we get one – a variable with a value of Master Noodle and this is the argument that gets passed to mkdir.

Exit Status

Fish will store the exit status of the previous command in $status, rather than in $ as it does in other shells:

> false

> echo $status

1

A non-zero output is a failure, while zero is a success

Exports

Fish also doesn't have the export command that other shells have. Instead, variables are exported using the options set, followed by either -x or –export:

> set -x MyVariable SomeValue

> env | grep MyVariable

MyVariablem=SomeValue

You can use -e or –erase to get rid of a variable

> set -e MyVariable

> env | grep MyVariable

(no output)

Lists

Above we used a command called set and this uses quotes to make sure that Master Noodle is treated as just one argument. If Master and Noodle had been separate arguments, name would have had a value of list length 2. In fish, all variables are classed as lists and they can contain no values or any number of values.

Some variables will have just one value, $PWD being one of them. Conventionally, we would talk about the value of that variable but, in reality, it is the only value it has. Other variables have several values, like $PATH. During the expansion of the variable, it will extend into several arguments:

> echo $PATH

/usr/bin /bin /usr/sbin /sbin /usr/local/bin

A list is not able to contain any other list; there is no such thing as recursion here. A variable contains a list of strings and that is it. If you wanted to get a list length, you would use count:

> count $PATH

5

You can append to any list by setting that list to itself with some extra arguments. In this example, we are going to append /usr/local/bin to $PATH:

> set PATH $PATH /usr/local/bin

Use square brackets to access each individual element. Indexing begins at 1 from the start and -1 from the end:

```
> echo $PATH
/usr/bin /bin /usr/sbin /sbin /usr/local/bin
> echo $PATH[1]
/usr/bin
> echo $PATH[-1]
/usr/local/bin
```

Fish also allows you to access slices, which are ranges of elements

```
> echo $PATH[1..2]
/usr/bin /bin
> echo $PATH[-1..2]
/usr/local/bin /sbin /usr/sbin /bin
```

And, just like with any other computer programming language, you can use for loops to iterate over lists:

```
> for val in $PATH
    echo "entry: $val"
  end
entry: /usr/bin/
entry: /bin
entry: /usr/sbin
entry: /sbin
```

entry: /usr/local/bin

Command Substitutions

A command substitution is when the output of a command is then used as an argument to a different command. Unlike many of the other shells, there are no backticks in fish for this; instead, parentheses are used:

> echo In (pwd), running (uname)

In /home/tutorial, running FreeBSD

One of the more common idioms is to use a variable to capture the output from a command:

> set os (uname)

> echo $os

Linux

A command substitution is not expanded inside quote marks. Instead, the quotes can be closed temporarily, the command substitution added in and then the quotes re-opened, all in one argument

> touch "testing_"(date +%s)".txt"

> ls *.txt

testing_1360099791.txt

Combiners

There is no special syntax in fish for combining commands; instead, there are three commands used – and, not, or

> cp file1.txt file1_bak.txt; and echo "Backup successful"; or echo "Backup failed"

Backup failed

Conditionals

Likewise, for the conditional execution of code, we use if, else and else if, depending on how a command is exited.

```
if grep fish /etc/shells
    echo Found fish
else if grep bash /etc/shells
    echo Found bash
else
    echo Got nothing
end
```

And we have the command, switch

```
switch (uname)
case Linux
    echo Hi Bix!
case Darwin
    echo Hi Hester!
case FreeBSD NetBSD DragonFly
    echo Hi Bobby!
case '*'
    echo Hi, stranger!
end
```

Functions

In fish, functions are lists of commands, which may or may not take arguments. We don't pass arguments as numbered variables in fish, like $1 but in single lists instead. There are built-in functions in fish which you can use to create functions:

```
> function say_hello
    echo Hello $argv
  end
> say_hello
Hello
> say_hello everybody!
Hello everybody!
```

And there is no special syntax for prompt or no aliases in fish. Functions have their own place and you can use the functions keyword to list the names of all functions, of which fish already has a large number:

Unlike other shells, fish does not have aliases or special prompt syntax. Functions take their place.

```
> functions
```

alias, cd, delete-or-exit, dirh, dirs, down-or-search, eval, export, fish_command_not_found_setup, fish_config, fish_default_key_bindings, fish_prompt, fish_right_prompt, fish_sigtrap_handler, fish_update_completions, funced, funcsave, grep, help, history, isatty, ls, man, math, nextd, nextd-or-forward-word, open, popd, prevd, prevd-or-backward-word, prompt_pwd, psub, pushd, seq, setenv, trap, type, umask, up-or-search,

vared

To find the source for a function, you pass its name to functions

```
> functions ls
function ls --description 'List contents of directory'
    command ls -G $argv
end
```

Loops

These are similar to other shells and to many computer programming languages:

While loops:

```
> while true
    echo "Loop forever"
end
Loop forever
Loop forever
Loop forever
...
```

For Loops:

For loops are used for iterating over lists, i.e. lists of files:

```
> for file in *.txt
    cp $file $file.bak
end
```

We use seq to iterate over lists of numbers:

```
> for x in (seq 5)
    touch file_$x.txt
end
```

Prompt

Fish does not have a prompt variable so, to display it, fish will execute a function that goes by the name of fish_prompt and the output from that function is the prompt. You are able to define your own if you want:

```
> function fish_prompt
    echo "New Prompt % "
end
New Prompt %
```

You can have multiple lines if you want and you can set the color using set_color, passing it as hex RGB or ANSI colors:

```
> function fish_prompt
      set_color blue
      date "+%m/%d/%y"
      set_color FF0
      echo (pwd) '>'
      set_color normal
  end
02/06/13
```

/home/tutorial >

If you run fish_config prompt, you can choose from several sample prompts

$PATH

$PATH is one of the environment variables in fish and it is home to the directories that fish uses for commands. $PATH is not delimited string; it is a list. If you wanted to prepend /usr/local/bin and /usr/sbin to $PATH, you would write:

> set PATH /usr/local/bin /usr/sbin $PATH

You can do this in config_fish or you can modify the universal variable, $fish_user_paths – this is prepended to $PATH automatically. For example, if you wanted to add /usr/local/bin permanently to $PATH, you would write:

> set -U fish_user_paths /usr/local/bin $fish_user_paths

The major advantage to doing it this way is that there is no need to mess around with files; all you do is run that once on the command line and the changes will happen in the current session and in future ones. NOTE – do NOT add that code line to config_fish – if you do, the variable will just grow in size every time you use fish!

Startup

When fish starts up it begins executing the commands in ~/.config/fish/config.fish. If that does not exist, you can create it. You can also create functions or variables directly in config_fish with the commands shown above, i.e.

> cat ~/.config/fish/config.fish

```
set -x PATH $PATH /sbin/


function ll
    ls -lh $argv
end
```

That said, the more efficient way to do it is to use universal variables and autoloading functions.

Autoloading Functions

When fish comes across a command, it will try to autoload a function for it, looking for a file that has the same name as the command in it. It will look in ~/.config/fish/functions/. For example, if you wanted a function called ll, you would add a text file called ll.fish to ~/.config/fish/functions:

```
> cat ~/.config/fish/functions/ll.fish
function ll
    ls -lh $argv
end
```

The would be the best way of defining a prompt:

```
> cat ~/.config/fish/functions/fish_prompt.fish
function fish_prompt
    echo (pwd) "> "
end
```

Universal Variables

Universal variables are those whose value is shared over all fish instances,

now and forever, even after rebooting. You can use set -U to make a universal variable:

> set -U EDITOR vim

Now in another shell:

> echo $EDITOR

Vim

Quick Quiz 5

1. What is fish?
2. How do you add a literal space into an argument?
3. What is the help command?
4. What sign is used for variable substitution?
5. What are the three combiners?
6. What are the three conditionals?
7. If syntax is highlighted in red, what does it mean?

Chapter 6: File and Directory Permissions

Code in this chapter is courtesy of
http://ryanstutorials.net/linuxtutorial/permissions.php

Now we are going to look at setting file and directory permissions. A permission is used to specify what a user can and cannot do with a directory or a file. This makes permissions very important for creating a working environment that is secure. You wouldn't want anyone to be messing about and making changes to your files and your system files also have to be kept safe from deliberate or accidental damage. Linux permissions are actually very easy to get to grips with.

Permissions dictate that you can read, write and/or execute specific files and directories. These are referred to by a single letter for each of them:

- r – read, allowing you to read file contents
- w – write, allowing you to edit the contents of a file
- x – execute, allowing you to run a file as if it were a script or a program

For each file, three sets of users are defined:

- owner – one user who is the file owner, usually the person who was responsible for creating the file but not always
- group – each file will belong to one specific group
- others – anyone who doesn't fall into one of the above two sets

So, there are three permissions and three sets of users and that, in a nutshell, is what permissions are. Now let's look at how to see and to edit them:

Viewing Permissions

To view the permissions on a file, we use the command, ls, with the long listing option:

ls -l [path]

ls -l /home/ryan/linuxtutorialwork/frog.png

-rwxr----x 1 harry users 2.7K Jan 4 07:32
/home/ryan/linuxtutorialwork/frog.png

In this example, to identify the permissions we are looking at the first 10 characters in the output.

- Character 1 tells you what the file type is. If it is d, it is a directory, – and it is an ordinary file
- The next 3 tell you what the file permissions are for the file owner. A letter tells you that there is a permission and identifies which one and the – is telling you that there are no permissions. In the above example, the owner of this file has permissions to read, write to and execute the file.
- The next 3 tell you which permissions a group has and, for this one, the group can read the file but cannot write to it or execute it. Note that the permissions always go in the same order – read, write, execute
- The last 3 characters are representative of the permissions for everyone else, or others. For this, they only the permissions to execute the file, not to read or write to it.

Changing Permissions

If you want to change the file or directory permissions, a command named chmod is used. Chmod stands for change file mode bits, something of a mouthful but really, all you need to remember is the mode bits as these are the permission indicators:

chmod [permissions] [path]

The permissions arguments for chmod have three separate components:

- Who the permission is being changed for – ugoa – User or owner, group, others and all
- Is the permission being granted or being revoked? A + or a – sign is used to indicate this
- Which permission is being set? - rwx – read, write, excite

Look at these examples:

First, we grant the group execute permission and then we take write permission away from the owner:

ls -l frog.png

-rwxr----x 1 harry users 2.7K Jan 4 07:32 frog.png

chmod g+x frog.png

ls -l frog.png

-rwxr-x--x 1 harry users 2.7K Jan 4 07:32 frog.png

chmod u-w frog.png

ls -l frog.png

-r-xr-x--x 1 harry users 2.7K Jan 4 07:32 frog.png

If you don’t want to go about the business of individually assigning permissions, you can do multiple ones at the same time:

ls -l frog.png

-rwxr----x 1 harry users 2.7K Jan 4 07:32 frog.png

chmod g+wx frog.png

ls -l frog.png

-rwxrwx--x 1 harry users 2.7K Jan 4 07:32 frog.png

chmod go-x frog.png

ls -l frog.png

-rwxrw---- 1 harry users 2.7K Jan 4 07:32 frog.png

So, it probably seems a little strange that the owner of a file can take away their own ability to read, to write to or to execute a file but there are some good reasons why this might happen. You could have a file that has data in it that you don't want accidentally changed for example. While these permissions can be removed, the owner may not remove their ability to set the permissions and that leaves control of the file with the owner.

The Shorthand Method of Setting Permissions

While the above method for setting permissions isn't difficult, it can be a little long-handed if you need to set specific permissions to specific files on a regular basis. For that, we have a shorter version of doing it. To understand this, you need to understand the number systems in Linux. The normal number system is a decimal system with a base 10 number – numbers 0 to 9. We also have the octal number system which is base 8 system – 0 to 7. There are three permissions and each is either on or it is off and, as such, there are 8

combinations. We can also use binary to represent the numbers and there are only two options with this – o and 1. The following shows you how octal numbers are mapped to binary:

Octal	**Binary**
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

What is interesting is that we can use just three binary values to represent the 8 octal numbers, with every possible combination of the 1 and 0 included. So, there are three bits and there are three permissions. 1 can represent on and 0 can represent off and that means that one octal number can be used as a way of representing a permission set for a set of users. If we use three numbers, we can represent permissions of all three - owner/user, group, and others. Let's look at a few examples:

ls -l frog.png

-rw-r----x 1 harry users 2.7K Jan 4 07:32 frog.png

chmod 751 frog.png

ls -l frog.png

-rwxr-x--x 1 harry users 2.7K Jan 4 07:32 frog.png

chmod 240 frog.png

ls -l frog.png

--w-r----- 1 harry users 2.7K Jan 4 07:32 frog.png

Permissions for Directories

The above permissions were for files but ewe can use the same ones for directories. However, they will behave a little differently:

- r – read, give the ability to read a directory's contents
- w – write, gives the ability to create directories and files in the directory
- x – execute, gives you the ability to get into a directory

Let's have a look at how these work:

ls testdir

file1 file2 file3

chmod 400 testdir

ls -ld testdir

-r-------- 1 ryan users 2.7K Jan 4 07:32 testdir

cd testdir

cd: testdir: Permission denied

ls testdir

file1 file2 file3

chmod 100 testdir

ls -ld testdir

---x------ 1 ryan users 2.7K Jan 4 07:32 testdir

cd testdir

ls testdir

ls: cannot open directory testdir/: Permission denied

Note that, when we ran ls on lines 5 and 14, we used the -d which is used to reference directory. Normally, if ls is given a directory as an argument, the contents of the dictionary will be listed. However, in this case, what we want to see is the direct permissions for that directory and using the -d option lets us do that.

These permissions might seem somewhat confusing to start with but all you need to remember is they are only for the directory, not for the files that are contained inside. For example, you could have a directory that you don't have permission to read even though it has files in it that you do have permission to read. Provide you are aware that the file is there and you know what its name is, you are still able to read it.

ls -ld testdir

--x------- 1 ryan users 2.7K Jan 4 07:32 testdir

cd testdir

ls testdir

ls: cannot open directory .: Permission denied

cat samplefile.txt

Kyle 20

Stan 11

Kenny 37

The Root User

On Linux, normally we would have two users who could change the file or directory permissions – the owner and the root or superuser. The root user is able to do everything in the system like a system administrator can. Normal users can generally only access directories and files that are held in their home directories and perhaps one or two others for collaboration and sharing purposes – this how the Linux system is kept stable and secure.

Basic Security

Your space on the Linux system is the home directory and it's up to you to make sure that's the way it stays. Some users will give themselves full permissions for read, write, and execute but will give no permission for others or groups. However, everyone has their own setup.

For the best security, you should not give group or others any write access to your own home directory but you can give execute permissions without adding

on read permissions. If you do this, other users can access your home directory but they cannot see what is there and a prime example of this would be a personal web page. Some systems run web servers and give each of their users a web space of their own; the most common setup is to create a directory in the home directory and call it public_html, allowing the server to read it and to display what is in it. However, because the web server is not the same user as you are, it cannot get in and it can't read the files. In this situation, it is necessary for permission to execute to be granted on the home directory, allowing the web server user to get to the resources it needs.

Exercises

Have a look at the permissions you have on your home directory and then explore the files inside and see what their permissions are

Next, change some of the permissions, using long and shorthand methods, and both absolute and relative paths, just so you get the hang of them. As a start, remove the read permission from one file and then try reading it to see what happens.

Next, create a new directory and add some files in. Now have a go at removing permissions from you on the directory and see what can and can't be done

Lastly, just explore the system and have a look at permissions in system directories

Quick Quiz

1. Which permission, when applied to a directory in the file system, will allow a user to enter the directory?
    - Read
    - Write
    - Execute
    - Access Control
2. A user needs to open a file, edit it, and then save the changes. What permissions does he need to do this? (Choose two.)
    - Read
    - Write
    - Execute
    - Modify
3. A file named staff.odt has a mode of rw-r- -r- -. If rbob is the file's owner, what can he do with it?
    - He can open the file and view its contents, but he can't save any changes.
    - He can open the file, make changes, and save the file.
    - He can change ownership of the file.
    - He can run the file if it's an executable.
4. A file named staff.odt has a mode of rw-r- -r- -. If mhuman is not the file's owner but is a member of the group that owns this file, what can he do with it?
    - He can open the file and view its contents, but he can't save any changes.
    - He can open the file, make changes, and save the file.
    - He can change ownership of the file.
    - He can run the file if it's an executable.

Chapter 7: Getting to Grips with Editors

The code in this chapter is courtesy of
http://ryanstutorials.net/linuxtutorial/vi.php

There are so many text editors to choose from for Linux that it can easily become confusing. In this chapter, we are going to look at nano and vi, two very simple editors for Linux.

Nano

Nano is already installed in many of the Linux distributions and it works very well with Sudo. Here's an overview of how to use it.

Running Nano

There are two ways to run nano:

- At the command prompt, type in nano
- Or input the following command - nano/name of file – insert the name of the file and nano will look for it; if it finds it, it will open it, if not you will get an empty buffer with the filename as its name

At the top of the screen you will see information about the file you are working on or, if none, it will say New Buffer. Following that is the content of your document in the form of text. Then is a system message that gives you information about the program that is executing a function and it will likely say "new file". Lastly, you will see the shortcut lines.

Nano is a WYSIWYG editor, or "what you see is what you get". Unless you used a key to modify your text, what you type in is what goes into the text input. Type something into it now so that we have something to play around with.

Shortcuts

Shortcuts are program functions, like exiting, saving, etc. and you will see the commonest ones at the bottom of the screen. You do not need to use SHIFT when you are using the shortcuts and they are all in lowercase and unmodified number format. If you press on CTRL+G you will see a list of all the shortcuts you can use; have a look and then press CTRL+X to come out of it.

Saving a file is known as "writing out" and you do this by pressing CTRL+O. Next, you are asked for a name and the shortcuts will change to show what you can input to finish this command.

To insert the contents from another file into the one you are working on, type in CTRL+R and to cancel both commands, simply press CTRL+C. If you have any trouble using the CTRL key just press ESC twice. And, if you see commands that require you to use the Meta key, this is the Alt key on your keyboard.

To exit nano, press CTRL+X

Navigating Nano

To navigate a text file you can use the arrow keys, home, page up, page down and end keys if you want but there is also another way:

- CTRL+F – move the cursor forward
- CTRL+B – move the cursor backward
- CTRL+P – move the cursor up one line
- CTRL+N – move the cursor down one line
- CTRL+A – takes the place of the Home key
- CTRL+E – takes the place of the End key
- CTRL+V – moves down one page
- CTRL+Y – moves up one page

- CTRL+SPACE – move forward one word at a time
- ALT+SPACE – move backward one word at a time
- CTRL+C – indicates where the cursor is at any given time

Copy, Cut and Paste

To copy text, you move the cursor to where you want to start and highlight it using CTRL+^ all text between the start point and the cursor will be highlighted.

- ALT+^ - copies the highlighted text
- CTRL+K – cuts the highlighted text
- CTRL+U – pastes the text where you specify with your cursor

To delete a whole line, use CTRL+K but do not highlight the text

Working with Vi

Vi is also a command line editor and is a very powerful one to boot. It is a plain text editor, like Notepad on Windows and, to use it, you use only your keyboard, not the mouse. Vi has two operating modes – Insert or Input mode and Edit. In the Insert mode, you can insert content into the current working file and in Edit mode, you navigate around the file, performing things like search, copy, delete, replace, save etc. To run vi, simply type one command line argument containing the file name that you want to edit:

vi <name of file>

if you don't specify a filename, you can open it from within but it is much easier to simply close vi down and start again. Rather than going into reams of explanations about how vi works, I am going to simplify it with some instructions for things to type in. Go ahead and open the Linuxstuff directory you created earlier.

In vi, type in

vi firstfile

If you have a file called firstfile, it will be opened; if not, vi will create it for you and it will then open it. You should now see something that looks like this:

~

~

"firstfile" [New File]

Because you are in edit mode, you now need to enter Insert mode, just by pressing i on your keyboard:

~

~

~

~

~

-- INSERT --

Type in a few lines of text and then press the Esc button; this puts you back in Edit mode

Save and Edit

You can do this in a few ways but they all do the same so go with what works for you. First, ensure you are still in Edit mode then use one of these:

- ZZ (use capital letters) – save and exit
- :q! – exit, discarding any changes made since the file was last saved

- :w – save but do not exit
- :wq – save and exit

Most vi commands will be expected as soon as the key sequence is input. However, any command that starts with the colon requires that you press enter after inputting the command.

Save the file that you are working on now and exit out of it.

Viewing Files

vi is, essentially, a text editor but you can view files with it as well. However, there are two more commands that you can use which are better. The first is cat, short for concatenate and it is used for joining files together; in its basic form, you can view files with it. Type in the following using the name of the file you just created.

cat<filename>

you should now see the contents of that file on your screen followed by a prompt. This will only happen if you run cat with a single argument. If you run it without any arguments, the cursor will move on to the following line and nothing else will happen.

This command is fine when you don't have a large file but if you do, you won't be able to read the contents very easily so we use a different command; less

less <filename>

This lets you move around a file using the up and down arrow keys or by using the spacebar to move on a page or pressing b to go back. Pressing q will exit you out of the file.

less allows you to move up and down within a file using the arrow keys. You may go forward a whole page using the **Spacebar** or back a page by

pressing **b**. When you are done, you can press **q** for quit. Tyr this on the file you created.

Navigating in vi

Open the file again and go into insert mode. Enter a couple of paragraphs and then press Esc so you go back into edit mode. To navigate around the file, use the following commands. Do this now and see how they work:

- The arrow keys – move the cursor
- j – move the cursor down
- k – move the cursor up
- h – move the cursor right
- l – move the cursor left
- $ - move the cursor to the end of the line you are on
- ^ - move the cursor to the start of the line you are on
- nG – move to the nth line, for example, 6G will take you to line 6
- G – move to the final line in the text
- w – move to the start of the next word
- nw – move forwards an amount of words, for example, 4w will move forward 4 words
- b – go back to the start of the last word
- nb – go back to the specified word
- {- - go back a paragraph
- }- go forward a paragraph

Delete Content

There are loads of ways to navigate vi and some of then, the n commands, let you specify a number. Deleting works the same way and there are a few delete commands that let us add in movement commands so that we can specify what is to be deleted. Using your open file, play around with the following delete

commands and see how they work:

- x – deletes one character
- nx – deletes the number of characters specified
- dd – deletes the line you are on
- dn – delete to where the n command would take you, i.e. d6w will delete 6 words

Undo

To undo a change in vi, including a deletion, is easy:

- u (lower case) – undoes the previous action and you can keep pressing it to undo several actions
- U (capital) – undoes all the changes to the line you are on

Exercises

1. Create a new file and add some content to it
2. Save the file and then try to view it using cat and less
3. Access the file in vi and add more content
4. Use all the movement commands to navigate around your file
5. Play with the delete commands, seeing how they all work
6. Undo one or more of the changes you have made

Quick Quiz

1. How do you run nano?
2. How do you bring up a list of shortcuts in nano?
3. When navigating text in Nano, how do you move forward one word at a time?
4. How do you go form Edit mode to Insert mode in vi?
5. How do you save and exit a file in vi?
6. Which two commands are used for viewing files in vi?

Chapter 8: Environment Variables

The code in this chapter is courtesy of https://wiki.archlinux.org/index.php/environment_variables

Environment variables are objects that are given names and contain data that is used by at least one, usually more applications. Put simply, an environment variable is nothing more than a variable that has a name and a value. Environment variable values could be the default editor that is to be used, the location of all files that can be executed in the system or the locale settings for the system, for example. Newbies to Linux might find this a bit cumbersome but the environment variable is a very good way of sharing the configuration settings between several processes and applications.

Utilities

The package called coreutils is home to the printenv and env programs. In order to list all the current environment variables that have a value, you would simply type:

```
$ printenv
```

Some of these variables are user-specific and you can check them by comparing what is output from printenv as a root user and a normal user.

The utility called env is used to run commands in an environment that has been modified. The following code example runs xterm with EDITOR, the environment variable, set to vim. This will have no effect on the global version of the same environment variable:

```
$ env EDITOR=vim xterm
```

Each different process will store their own environments in a file called /proc/$PID/environ. Contained in this file will be key value pairs, delimited with null characters (\x0).

Defining a Variable

Global variables

Most of the distributions of Linux advise you to change or add the variable definitions in /etc/profile or in another location. However, you do need to bear in mind that some configuration files are package specific and these contain some of the variable settings, like /etc/locale.config. These environment variables must be managed and maintained and you must pay special attention to all the files that may contain an environment variable. In principle, any of the shell scripts can initialize an environment variable but, under the traditional conventions, those statements should be only in some files. The following are the files that are to be used for defining a global environment variable:

- /etc/environment, /etc/profile and any shell-specific configuration file. These all have their own set limitations so only choose the right one for the purpose it is needed for
- /etc/environment – the pam_env module uses this and this is shell-agnostic. This means that you cannot use glob expansion or scripting. This file will only take variable-value pairs.

In the next example, we are adding ~/bin directory to the PATH for the respective user. All you do is add this line to the configuration file of the chosen environment variable (/etc/profile or /etc/bash.bashrc):

If the user ID is greater than or equal to 1000 & if ~/bin exists and is a directory & if ~/bin is not in your $PATH

```
# then export ~/bin to your $PATH.

if [[ $UID -ge 1000 && -d $HOME/bin && -z $(echo $PATH | grep -o $HOME/bin) ]]

then

    export PATH="${PATH}:$HOME/bin"

fi
```

Per user

Note - the user instance of systemd and the dbusdaemon will not inherit the variables that are in places such as ~/.bashrc etc. As such, any program that is dbus activates, like Gnome Files, will not use them unless told to.

It isn't always the right thing to define environment variables globally. On occasion, for example, you may want to add /home/my_user/bin to the variable called PATH but not for all the users. This is where locally defined variables come in and these can be defined in many files:

- ~/.pam_environment is the user equivalent of /etc/security/pam_env.conf, used by pam_env module.
- ~/.profile is used by several shells as a fallback.

If you want to add a directory for local use to PATH, add the following into the ~/.bash_profile:

```
export PATH="${PATH}:/home/my_user/bin"
```

If you want to update that variable, you must either log back in or source: $ source ~/.bash_profile.

Graphical applications

If you want to use an environmental variable for a graphical user interface

application, your variables can be placed in xinitrc or, if you are using a display manager, in xprofile. Look at this example:

~/.xinitrc

export PATH="${PATH}:~/scripts"

export GUIVAR=value

Per session

On occasion, you may even need definitions that are much stricter. For example, you may want to run an executable file from a specified directory on a temporary basis without needing to input the absolute path for each one or without having to edit the shell config files just for the short time needed. In this case, you would define the PATH variable in the session you are currently in and combine with the command for export. Provided you stay logged in, the PATH variable will continue to use the temporary settings. If you wanted to add a directory that is session-specific to PATH, you would type:

$ export PATH="${PATH}:/home/my_user/tmp/usr/bin"

Examples

In this section, we are going to look at some of the more common of the Linux environment variables and talk about their values.

DE

This is indicative of Desktop Environment, specifically the one in use. xdg-open uses DE to pick a file-opener application that is user-friendly than the one provided by DE. Some packages must also be installed using DE and the recognized values of the variable are:

- gnome
- kde

- lxde
- xfce
- mate

The variable has to be exported before the window manager can be started, for example:

~/.xinitrc

export DE="xfce"

exec openbox

This forces xdg-open to use exo-open, which is more user-friendly, because it automatically assumes it is in xfce. For configuration, exo-preferred applications should be used.

- **DESKTOP_SESSION**

This is like DE but is used inside the LXDE environment. For example, when DESKTOP_SESSION has been set to LXDE, xdg-open will automatically use the pcmanfm associations for files.

- **PATH**

PATH is home to a list of directories, separated by colons, and this is where the system will look for any executable files. When the shell interprets a regular command, such as ls, it will look for any executable file that has the same name as the command that is in the listed directories and it will then execute it. If you want to run executables that have not been put into PATH, you need to use the absolute path to the specific executable.

- **HOME**

This is where you will find the path that leads to the home directory of the user currently logged in. HOME can be used by several applications as a way of

associating configuration files with the user that is currently running the variable.

- **PWD**

This is where you will find the path that leads to the working directory

- **OLDPWD**

This is where you will find the path that leads to the previous working directory, i.e. the value that PWD had before the last execution of cd

- **SHELL**

This is where we find the path that leads to the preferred shell of the user. This may not be the shell that is currently in use but the variable is set by Bash on startup

- **TERM**

This contains the type of terminal running, for example, xterm-256color. This tends to be used by programs that run in the terminal and need capabilities that are terminal-specific.

- **PAGER**

This has the command that runs the program that is used to list file contents, i.e. /bin/less

- **EDITOR**

This has the command used for running the light program for file editing, i.e. /usr/bin/nano. For example, you could write a switch that is interactive between nano or gedit under X:

export EDITOR="$(if [[-n $DISPLAY]]; then echo 'gedit'; else echo 'nano'; fi)"

- **VISUAL**

This is where you will find the command that runs the full editor used for tasks that are somewhat more demanding, like mail editing

- **MAIL**

This is where you will find the location of incoming mail and the setting traditionally used is /var/spool/mail/$LOGNAME

- **BROWSER**

This is where you find the path that leads to the web browser. It is very useful for when you want an interactive configuration file set for the shell as it can be altered dynamically, dependent on how available a graphic environment is:

```
if [ -n "$DISPLAY" ]; then
   export BROWSER=firefox
else
   export BROWSER=links
fi
```

- **ftp_proxy and http_proxy**

These contain the HTTP and the FTP servers:

```
ftp_proxy="ftp://192.168.0.1:21"
http_proxy="http://192.168.0.1:80"
```

- **MANPATH**

This is where you find a list of directories, each separated by a colon and this is where man will search for man pages.

NOTE – in etc/profile, you will find a comment that says “Man is much better

than us at figuring this out" so leave this variable as a default, for example, /usr/share/man:usr/local/share/man

- **INFODIR**

This has a list of directories, separated by colons, and this is where the info command will look for info pages, for example, /usr/share/info:usr/local/share/info

- **TZ**

This is used for setting time zones that are different to the one used in the system. If you look in usr/share/zoneinfo, you can see zones that can be used as references

Using pam_env

The PAM module is used for loading variables that need to be set into the environment and these come from the following files:

- /etc/security/pam_env.conf
- /etc/environment
- ~/.pam_environment.

/etc/environment must be made up of VARIABLE=VALUE pairs, each on a separate line

/etc/security/pam_env.conf and ~/.pam_environment both have exactly the same format:

VARIABLE [DEFAULT=[value]] [OVERRIDE=[value]]

It is this format that allows for the expansion of variables that are already defined within other variable values, using ${VARIABLE}. @{SHELL} and @{HOME} are both special variables that will expand as per the definition in etc/passwd.

The following is an example of the configuration of a basic user directory:

~/.pam_environment

```
# Setting the variables that reuse your $HOME
XDG_CACHE_HOME   DEFAULT=@{HOME}/.cache
XDG_CONFIG_HOME  DEFAULT=@{HOME}/.config
XDG_DATA_HOME    DEFAULT=@{HOME}/.local/share

# You may reuse XDG_RUNTIME_DIR for runtime files
ICEAUTHORITY     DEFAULT=${XDG_RUNTIME_DIR}/ICEauthority

# You may reuse variables you have already defined
GNUPGHOME        DEFAULT=${XDG_CONFIG_HOME}/gnupg

# You may define variables as VARIABLE=VALUE pair
EDITOR=nano

# Same as above
EDITOR           DEFAULT=nano

#Incorrect: you may not reuse other variables in VARIABLE=VALUE pair
#GNUPGHOME=${XDG_CONFIG_HOME}/gnupg
```

#Incorrect: missing {}

#GNUPGHOME DEFAULT=$XDG_CONFIG_HOME/gnupg

Note: This file will be read first, before anything else, even ~/.{,bash_,z}profile and ~/.zshenv.

Quick Quiz

1. What does "../.." stand for ?
    - Current directory
    - Up one directory
    - Up two directories
    - None of Above
2. What does cd/ mean in Linux?
    - Current directory
    - Root directory
    - Up one directory
    - None of the above
3. Which variable gives the exit status of the last command executed in the shell?
    - $*
    - $!
    - $?
    - $@
4. How will you assign the value of variable var2 to var1?
    - Var1=var2
    - $var1=$var2
    - Var1=$var2
    - $var1=var2

Conclusion

Thank you again for purchasing this book!

I hope this book was able to help you to understand Linux and what it is all about, as well as how to use it.

The next step is to practice. There is plenty of information on the internet, plenty of Linux forums and lots of courses you can take, some paid, some free, all of which will take you deeper into Linux and teach you more about how it works and what you can do with it.

Finally, if you enjoyed this book, then I'd like to ask you for a favor, would you be kind enough to leave a review for this book on Amazon? It'd be greatly appreciated!

Thank you and good luck!

Answers

Quick Quiz 1

1. VirtualBox is a piece of software that provides you with a virtual machine, allowing you to run other operating systems on your computer without interfering with your main system
2. CentOS is one of the Linux distributions
3. ssh stands for Secure Shell
4. To access the BIOS settings, when your computer starts up press F1, F2, F12 or ESC, depending on your system

Quick Quiz 2

1. /bin
2. /boot
3. /root
4. /dev
5. /etc
6. /home

Quick Quiz 3

1. env
2. $$
3. Files that begin with ‘a’
4. A command interpreter

Quick Quiz 4

1. cd
2. clear
3. ls
4. pwd

Quick Quiz 5

1. A command line interpreter like BASH
2. / or ‘’ or “”
3. man
4. $
5. and, not, or
6. if, else, else if
7. Invalid command

Quick Quiz 6

1. execute
2. read and write permissions
3. he can open the file, make changes and save the file
4. he can open the file, and view the contents but cannot save the file

Quick Quiz 7

1. nano or nano/name of file
2. CTRL+G
3. CTRL+SPACE
4. i
5. ZZ
6. cat and less

Quick Quiz 8

1. up two directories
2. root directory
3. $?
4. Var1=$var2

References

https://fishshell.com

http://ryanstutorials.net/linuxtutorial/permissions.php

http://ryanstutorials.net/linuxtutorial/vi.php

https://wiki.archlinux.org/index.php/environment_variables