



BMKG

METEOROLOGICAL TRAINING MODUL

01.04

**Ubuntu Linux
Operation: Installation
Procedure and Basic
Command**

Authors :

Zainal Abidin

Thahir Daniel Hutapea

EDUCATION AND TRAINING CENTRE

THE AGENCY FOR METEOROLOGY CLIMATOLOGY AND GEOPHYSICS

2020

FOREWORD


Praised be to God Almighty, teaching materials about Ubuntu Linux Operation: Installation Procedure and Basic Command has been completed. Linux is the best-known and most-used open source operating system. Linux has been garnering strength for a number of years now, and its growing popularity drives more and more users to make the jump. Understanding the basics of linux is an important step in to carry out work stages in developing and using Numerical Weather Prediction. Hence, knowledge of basic linux is also the fondation in the context of enhancing weather forecast quality service, due to considering the need for quality human resources is becoming increasingly urgent amidst various global, regional and local weather conditions that are increasingly uncertain. The purpose of this modul is to help you discover and understanding basic command of linux operation. Within this frame of reference, the author tries to build teaching materials about basic linux, especially using the Ubuntu linux operation.

Nevertheless, future improvements are needed with the intention that these modules can be kept up-to-date and in tune with developments in science and technology. Hopefully this teaching material can be useful and I express my gratitude and high appreciation to the drafting team for their contribution and cooperation.

Jakarta, October 2020

HEAD OF EDUCATION AND TRAINING CENTRE

THE AGENCY FOR METEOROLOGY CLIMATOLOGY AND GEOPHYSICS



Maman Sudarisman

TABLE OF CONTENTS

CHAPTER I Linux and Open Source	5
1.1 What is Linux	5
1.2 What is Open Source	5
1.3 Linux and Open Source	5
1.4 What are the values of open source?	6
CHAPTER II Linux Installation	7
2.1 Virtual Machine	7
2.2 Linux Virtual Machine Installation	8
CHAPTER III Linux File and Directories Management	14
3.1 Linux File/Folder Structure.....	14
3.2 Work on the Command Line	16
3.3 Navigating the Filesystem.....	17
3.4 Linux Package Management	18
3.5 Basic file editing	21
3.6 Searching text using regular expression.....	24
3.6 Manage file permission and ownership	26
3.6.1 Change file permission and ownership	27
3.6.2 Change file ownership	28
3.7 Process text stream using filters.....	28
3.8 Use stream, pipes and redirection	30
3.8.1 Streams	30
3.8.2 Redirection	31
3.8.3 Pipes.....	33
3.9 Perform basic file management.....	34
CHAPTER IV Linux Processes Monitoring and Task Scheduling	36
4.1 Monitor and Kill Processes	36

4.2 Create Links	37
4.3 Find System Files.....	37
4.5 Automate Tasks by Scheduling Jobs	38
4.6 Make and install programs from source	38
CHAPTER V Linux Shell Programming.....	40
5.1 Working with Shell Scripting	40
5.1.1 Shell Environment	40
5.1.2 Write Simple Script.....	41
5.1.3 Variables.....	41
5.1.4 Shell Arguments.....	42
5.2 Shell Programming	42
5.3 Create simple script for downloading GFS.....	42
CHAPTER VI Summary and Quiz.....	47
6.1 Summary.....	47
6.2 Quiz.....	47
REFERENCES	48

CHAPTER I Linux and Open Source

Success Indicators	Participants can explain the concept of Open Source and its relation with linux
---------------------------	---

1.1 What is Linux

Linux is the best-known and most-used open source operating system. As an operating system, Linux is software that sits underneath all of the other software on a computer, receiving requests from those programs and relaying these requests to the computer's hardware.

How does Linux differ from other operating systems?

In many ways, Linux is similar to other operating systems you may have used before, such as Windows, macOS (formerly OS X), or iOS. Like other operating systems, Linux has a graphical interface, and the same types of software you are accustomed to, such as word processors, photo editors, video editors, and so on. In many cases, a software's creator may have made a Linux version of the same program you use on other systems. In short: if you can use a computer or other electronic device, you can use Linux.

But Linux also is different from other operating systems in many important ways. First, and perhaps most importantly, Linux is open source software. The code used to create Linux is free and available to the public to view, edit, and—for users with the appropriate skills—to contribute to.

Linux is also different in that, although the core pieces of the Linux operating system are generally common, there are many distributions of Linux, which include different software options. This means that Linux is incredibly customizable, because not just applications, such as word processors and web browsers, can be swapped out. Linux users also can choose core components, such as which system displays graphics, and other user-interface components.

1.2 What is Open Source

Open source is a term that originally referred to open source software (OSS). Open source software is code that is designed to be publicly accessible—anyone can see, modify, and distribute the code as they see fit.

Open source software is developed in a decentralized and collaborative way, relying on peer review and community production. Open source software is often cheaper, more flexible, and has more longevity than its proprietary peers because it is developed by communities rather than a single author or company.

Open source has become a movement and a way of working that reaches beyond software production. The open source movement uses the values and decentralized production model of open source software to find new ways to solve problems in their communities and industries.

1.3 Linux and Open Source

Linux is a free, open source operating system (OS), released under the GNU General Public License (GPL). It's also become the largest open source software project in the world. The

Linux operating system was created as an alternative, free, open source version of the MINIX operating system, which was itself based on the principles and design of Unix.

Because Linux is released under an open source license, which prevents restrictions on the use of the software, anyone can run, study, modify, and redistribute the source code, or even sell copies of their modified code, as long as they do so under the same license.

1.4 What are the values of open source?

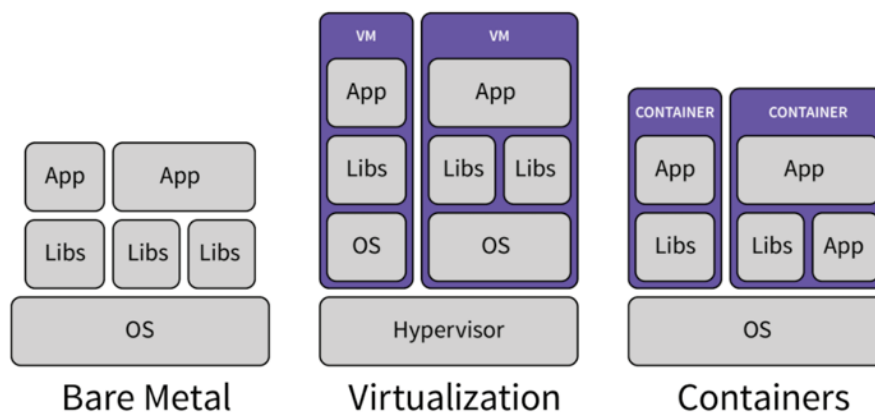
There are lots of reasons why people choose open source over proprietary software, but the most common ones are:

- Peer review: Because the source code is freely accessible and the open source community is very active, open source code is actively checked and improved upon by peer programmers. Think of it as living code, rather than code that is closed and becomes stagnant.
- Transparency: Need to know exactly what kinds of data are moving where, or what kinds of changes have happened in the code? Open source allows you to check and track that for yourself, without having to rely on vendor promises.
- Reliability: Proprietary code relies on the single author or company controlling that code to keep it updated, patched, and working. Open source code outlives its original authors because it is constantly updated through active open source communities. Open standards and peer review ensure that open source code is tested appropriately and often.
- Flexibility: Because of its emphasis on modification, you can use open source code to address problems that are unique to your business or community. You aren't locked in to using the code in any one specific way, and you can rely on community help and peer review when you implement new solutions.
- Lower cost: With open source the code itself is free—what you pay for when you use a company like Red Hat is support, security hardening, and help managing interoperability.
- No vendor lock-in: Freedom for the user means that you can take your open source code anywhere, and use it for anything, at anytime.
- Open collaboration: The existence of active open source communities means that you can find help, resources, and perspectives that reach beyond one interest group or one company.

CHAPTER II Linux Installation

Success Indicators	Participants can demonstrate linux installation virtual machine
---------------------------	---

There are many ways to do linux installation, i.e bare metal, virtual machine, containerization. The traditional way to install linux is bare metal, linux is installed in PC/Server and has full potential infrastructure resources. Virtual machine allows us to run (guest) OS inside the (host) OS, thanks to virtualization technology VT-X from Intel and AMD-V from AMD that create virtual processor, storage, and network. Container need container manager (i.e Docker), so far container manager only runs on linux.



All methods need ISO linux installation that can be downloaded from the official linux distribution or open source mirror servers which are mostly located in every countries. This module will use Ubuntu Linux 20.04 that can be downloaded from Ubuntu website <https://ubuntu.com/download/desktop>

2.1 Virtual Machine

Virtual machines allow you to run an operating system in an app window on your desktop that behaves like a full, separate computer. You can use them play around with different operating systems, run software your main operating system can't, and try out apps in a safe, sandboxed environment.

A virtual machine app creates a virtualized environment—called, simply enough, a virtual machine—that behaves like a separate computer system, complete with virtual hardware devices. The VM runs as a process in a window on your current operating system. You can boot an operating system installer disc (or live CD) inside the virtual machine, and the operating system will be “tricked” into thinking it’s running on a real computer. It will install and run just as it would on a real, physical machine. Whenever you want to use the operating system, you can open the virtual machine program and use it in a window on your current desktop.

In the VM world, the operating system actually running on your computer is called the host and any operating systems running inside VMs are called guests. In a particular VM, the guest OS is stored on a virtual hard drive—a big, multi-gigabyte file stored on your real

hard drive. The VM app presents this file the guest OS as a real hard drive. This means you won't have to mess around with partitioning or doing anything else complicated with your real hard drive.

Virtualization does add some overhead, so don't expect them to be as fast as if you had installed the operating system on real hardware. Demanding games or other apps that require serious graphics and CPU power don't really do so well, so virtual machines aren't the ideal way to play Windows PC games on Linux or Mac OS X—at least, not unless those games are much older or aren't graphically demanding.

In this course, we will use VMware Workstation Player to make linux VM. VMware Workstation Player can be downloaded in <https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html/>

2.2 Linux Virtual Machine Installation

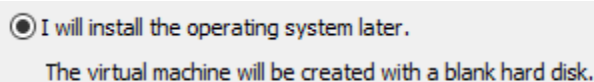
1. Open VMware Workstation Player
2. Click Create New Virtual Machine



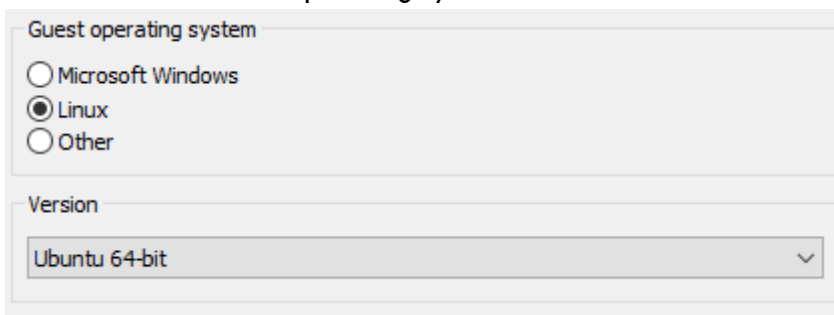
Create a New Virtual Machine

Create a new virtual machine, which will then be added to the top of your library.

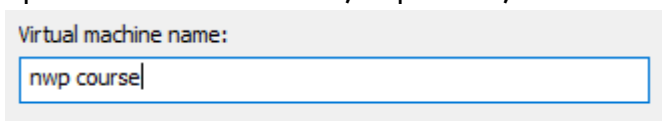
3. choose I will install the operating system later, click Next



4. choose Linux as Guest operating system and Ubuntu 64-bit version, click Next



5. input Virtual machine name, nwp course, click Next



6. input Maximum disk size (GB), 50 and choose Split virtual disk into multiple files, click Next

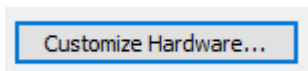
Maximum disk size (GB):

Recommended size for Ubuntu 64-bit: 20 GB

☐ Store virtual disk as a single file
☒ Split virtual disk into multiple files

Splitting the disk makes it easier to move the virtual machine to another computer but may reduce performance with very large disks.

7. click Customize Hardware



It allows us to define virtual hardware that want to be utilized. We can define number of processor, memory, network adapter and configuration for this VM.

8. choose Processors and choose 2 as minimum Number of processor cores

Device	Summary
Memory	2 GB
Processors	2
New CD/DVD (SATA)	Auto detect
Network Adapter	NAT
USB Controller	Present
Sound Card	Auto detect
Printer	Present
Display	Auto detect

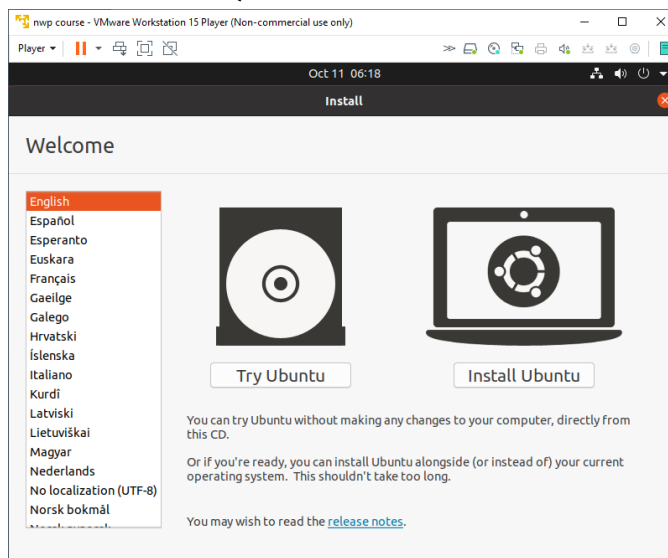
Processors

Number of processor cores:

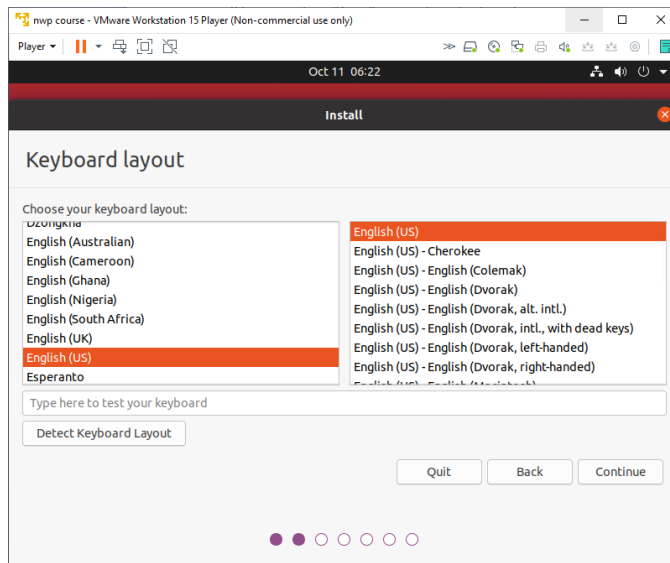
Virtualization engine

☐ Virtualize Intel VT-x/EPT or AMD-V/RVI
☐ Virtualize CPU performance counters
☐ Virtualize IOMMU (IO memory management unit)

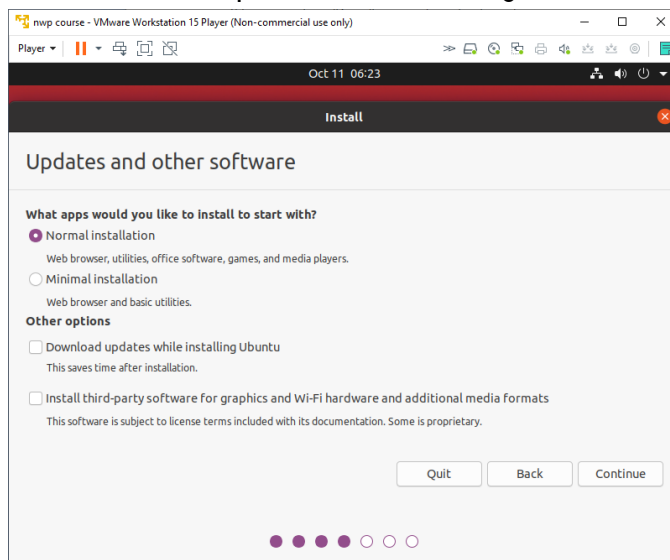
9. installation wizard, click Install Ubuntu



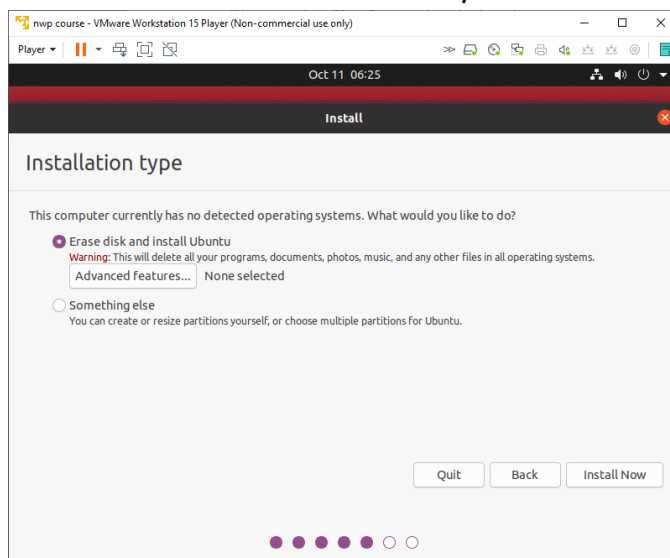
10. choose Keyboard setting



11. untick Download updates while installing Ubuntu

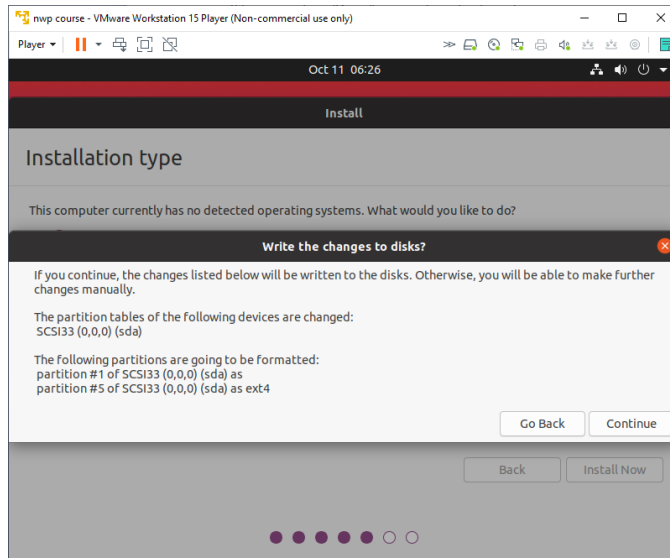


12. choose Erase disk and install Ubuntu, click Install Now

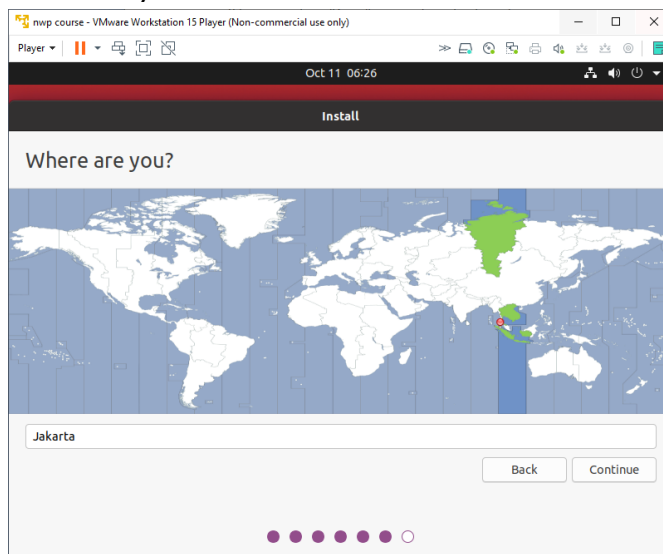


This option will simplify folder mounting configuration for the installation. If we chose “Something Else”, it is possible to configure the mounted folder and storage size and configuration. Information related with folder and mounting are explained in Chapter 3.

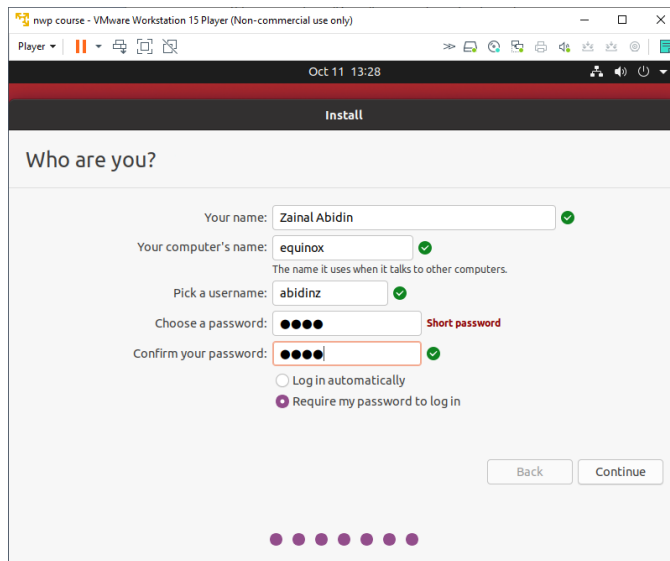
13. confirm that wizard will write changes to disks, click Continue



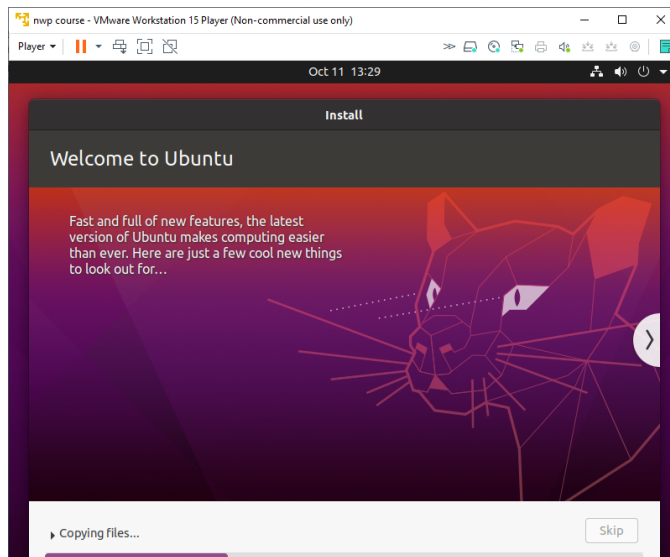
14. choose City location



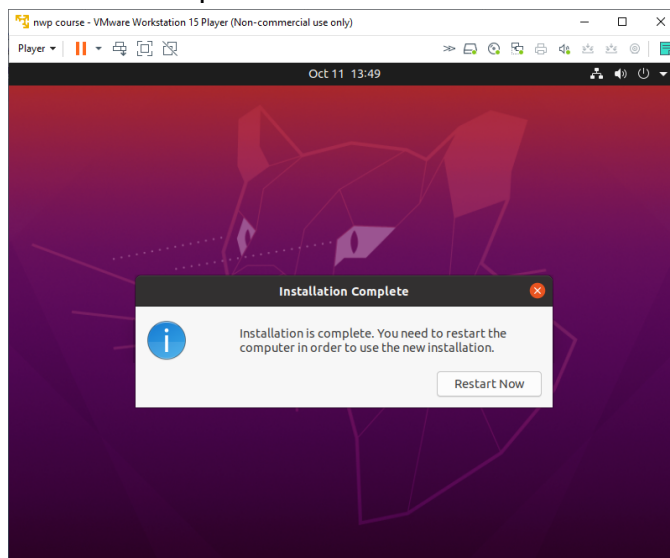
15. input user identity and name the machine, click Continue



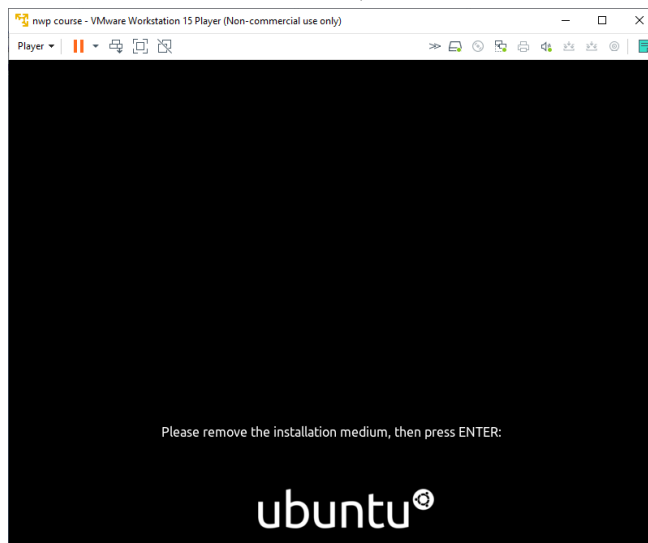
16. Linux installation begins



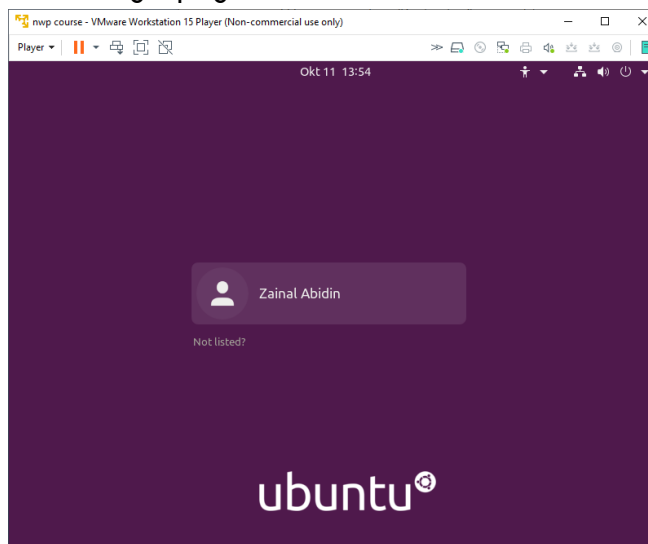
17. Installation complete



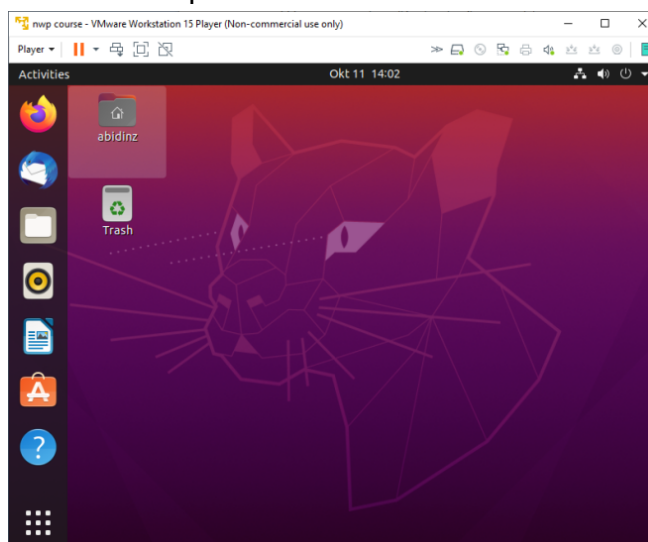
18. Remove installation medium, reboot VM



19. Ubuntu login page



20. Ubuntu desktop



CHAPTER III Linux File and Directories Management

Success Indicators

The Participants should be able to use basic linux commands to manage file and directories

3.1 Linux File/Folder Structure

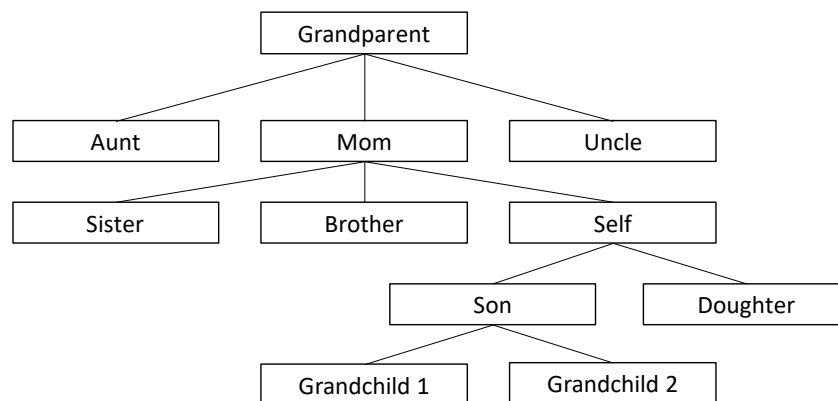
The filesystem is a set of data structures that usually resides on part of a disk and that holds directories of files. Filesystems store user and system data that are the basis of users' work on the system and the system's existence.

In this chapter the organization and terminology of the linux filesystems will be discussed, defines ordinary and directory files, and explains the rules of naming them. It also shows how to create and delete directories, move through the filesystem, and use absolute and relative pathnames to access file in various directories.

The Hierarchical Filesystem

Family tree

A hierarchical structure frequently takes the shape of a pyramid. One example of this type structure is found by tracing family's lineage; this hierarchical structure is called a family tree.

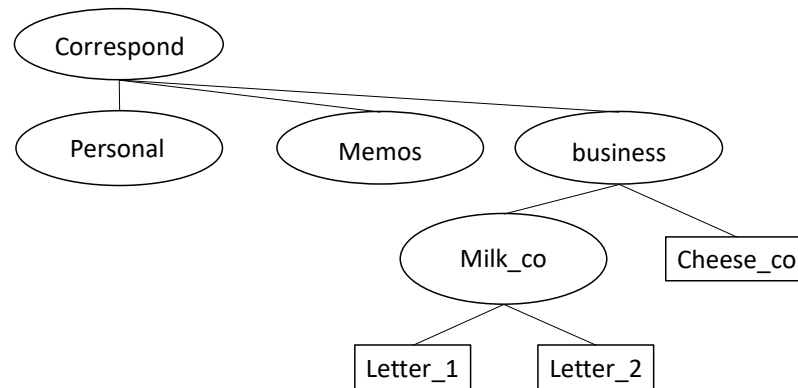


Directory tree

Like the family tree it resembles, the linux filesystem called a tree. It consists of a set of connected files. This structure allows you to organize files so you can easily find any particular one. On a standard linux system, each user starts with one directory, to which the user can add subdirectories to any desired level. By creating multiple levels of directories, a user can expand the structure as needed.

Subdirectory

Typically, each subdirectory is dedicated to a single subject, such as a person, project, or event. The subject dictates whether a subdirectory should be divided further.



One major strength of the linux filesystem is its ability to adapt to users' needs. You can take the advantage of this strength by strategically organizing your file so they are most convenient and useful.

Linux has different folder structure compared with Windows which has C: as root directory of the system. Linux has / (root) as root directory, all devices such as storage, optical drive, gpu card and other hardware or devices are represented with file. Below are common folder structure in linux, linux distributions has their own folder structure :

- / (root) It is the main folder structure in linux
- /bin user binaries, common user's binaries files
- /sbin system binaries, root user's binaries files
- /etc configuration files
- /dev device files
- /proc kernel and process information
- /var variable files, it contains logs and databases. In most cases, this folder can grow due to logs of the system
- /usr user programs
- /home user's home directory
- /boot boot files, contains boot loader file and linux kernel
- /lib shared libraries
- /media /mnt /mount mounted folder for storage
- /opt optional or additional software packages

One major strength of the linux filesystem is its ability to adapt to users' needs. You can take the advantage of this strength by strategically organizing your file so they are most convenient and useful.

FILENAME

Every file has filename. The maximum length of a filename varies with the type of filesystems; Linux supports several types of filesystems. Although most of today's filesystems allow files with names up to 255 characters' long, some filesystems restrict filenames to fewer character. While you can use almost any character in a filename, you will avoid confusion if you choose characters from the following list;

- Uppercase letters (A-Z)
- Lowercase letters (a-z)
- Numbers (0-9)
- Underscore (_)
- Period (.)

Comma (,)

Like children of one parent, no two file in the same directory can have the same name. file in different directory can have same filename.

3.2 Work on the Command Line

When we talk about the command line, we are really referring to the shell. The shell is a program that takes keyboard command and passes them to the operating system to carryout. The shell executes a program when you give it a command in response to its prompt.

The lines that contains the command, including any argument, is called the command line.

To write a command in a shell we need a program called terminal emulator to interact with the shell. A number of other terminal emulators are available for Linux, but they all do basically the same thing: give us access to the shell.

```
$
```

The first time we launch terminal emulator we should see something what is called the shell prompt, and its appear whenever the shell is ready to accept input. If the last character of the prompt is a hash mark (#) rather than a dollar sign, the terminal session has superuser privileges. This means that either we are logged in as the root user or we've selected a terminal emulator that provides superuser (administrative) privileges.

After seeing the shell prompt and its appear in normal condition, lets try some typing. Try to enter some gibberish at the prompt;

```
$ ksjhfrkjhs
ksjhfrkjhs: command not found
$
```

Since the command makes no sense; the shell tell us so and gives us another chance.

If we press the up-arrow key, we see that the previous command ksjhfrkjhs reappears after the prompt. This is called command history. Most Linux distributions remember the last 500 commands by default. Press the down-arrow key, and the previous command disappears.

Recall the previous command with the up-arrow key again. Now try the leftand right-arrow keys. See how we can position the cursor anywhere on the command line? This makes editing commands easy.

```
$ command [options] [arguments]
```

command is aprogram that tells the linux system to do something

options are generally preceded by a hypen (-), and for most commands, more than one option cen be strung together

```
$ command -[option1][option2]
$ command -option1-option2
```

Example:

```
$ ls -alR
$ ls -a -l -R
```

Try some simple commands.

We have learned to type, now let's try a few simple commands. The first command is date, this command display current time and date.

```
$ date
Wed Oct 7 23:16:18 WIB 2020
$
```


A related command is `cal`, which by default display calendar of the current month.

```
$ cal
October 2020
Su Mo Tu We Th Fr Sa
      1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

To see the current amount of free space on the disk drives, type `df`, likewise to display the free amount of free memory type `free` command;

```
$ df
Filesystem      1K-blocks    Used Available Use% Mounted on
rootfs          498799612 130359476 368440136  27% /
none            498799612 130359476 368440136  27% /dev
none            498799612 130359476 368440136  27% /run
none            498799612 130359476 368440136  27% /run/lock
none            498799612 130359476 368440136  27% /run/shm
none            498799612 130359476 368440136  27% /run/user
tmpfs           498799612 130359476 368440136  27% /sys/fs/cgroup
C:\             498799612 130359476 368440136  27% /mnt/c

$ free
              total        used        free      shared  buff/cache   available
Mem:      16586484      5280236      11076896        17720       229352     11172516
Swap:      50331648         2608       50329040
```

To end a terminal session can done by either closing the terminal window or entering the `exit` command at the shell prompt.

```
$ exit
```

3.3 Navigating the Filesystem

As a Linux user, one of the first skills that you need to master is navigating the Linux filesystem. These basic navigation commands will get you up to speed. In most cases, using linux command is easier than GUI.

```
pwd
```

it shows the current/working directory, telling where you are currently located in filesystem

Example:

```
$ pwd
/home/nwp
```

By default, linux terminal session starts from home folder, which in most cases home folder of the current user is located in `/home/<username>`. In this example, username is `nwp` and has home folder located in `/home/nwp`

```
ls
```

list directory contents and information about the files, by default sorted alphabetically.

Example:

```
$ ls
```

List the directory contents

```
$ ls -l
```

shows files in list with file information details

```
$ ls -lt
```

shows files in list with details, sorted by time

```
$ ls -ltr
```

shows files in list with details, sorted by time in reverse order

```
cd
```

Very closely related to the `pwd` command is the `cd` command. Changing directories is a frequent activity on a Linux system. As stated before, when you first log in, you're placed into your home directory. Every user on a Linux system has a home directory. Regular user accounts have personal directories under the `/home` directory. Your home directory is under `/home/<username>`. To view all user's home directories, `cd` to the `/home` directory.

Example:

```
$ cd
```

Change directory to home folder

```
$ cd /usr/bin
```

Change directory to `/usr/bin`

```
$ pwd
```

```
/usr/bin
```

Show current working directory

```
$ cd ~
```

Back to home folder, tilde (`~`) sign represents home folder of current user

```
$ cd Downloads
```

or

```
$ cd ~/Downloads
```

Change directory relative to Downloads, or `/home/<USERNAME>/Downloads`

```
mkdir
```

make a new directory

```
$ mkdir grib
```

make grib directory

```
$ mkdir forecast gfs ecmwf
```

make multiple directories, forecast gfs ecmwf directories

```
$ mkdir -p forecast/2020/10/05
```

make forecast/2020/10/05 directory recursively

```
rmdir
```

delete empty directory

Example:

```
$ rmdir gfs
```

Delete gfs directory, this directory has to be empty before deleted

3.4 Linux Package Management

Package management is a method of installing and maintaining software on the system. Nowadays, most people can satisfy all of their software needs by installing packages from their Linux distributor. This contrasts with the early days of Linux, when one had to download and compile source code in order to install software. Not that there is anything wrong with compiling source code; in fact, having access to source code is the great wonder of Linux. It gives the ability to examine and improve the system. It's just that working with a precompiled package is faster and easier.

In this chapter, we will look at some of the command-line tools used for package management. While all of the major distributions provide powerful and sophisticated graphical programs for maintaining the system, it is important to learn about the command-line programs, too. They can perform many tasks that are difficult (or impossible) to do using their graphical counterparts.

Packaging System

Different distributions use different packaging systems, and as a general rule a package intended for one distribution is not compatible with another distribution. Most distributions fall into one of two camps of packaging technologies; the Debian `.deb` camp and the Red Hat `.rpm` camp.

Packaging System	Distributions (Partial listing)
Debian Style (.deb)	Debian, Ubuntu, Xandros, Linspire
Red Hat Style (.rpm)	Fedora, CentOS, Red Hat Enterprise Linux, openSUSE, Mandriva, PCLinuxOS

Linux system provided by the distribution vendor in the form of package files, and the rest is available in source code form, which can be installed manually. The basic unit of software in a packaging system is the package file. A package file is a compressed collection of files that comprise the software package. A package may consist of numerous programs and data files that support the programs. Package files are created by a package maintainer. The package maintainer gets the software in source code form from the upstream provider (the author of the program), compiles it, and creates the package metadata and any necessary installation scripts. The package maintainer will apply modifications to the original source code to improve the program's integration with the other parts of the Linux distribution. Packages are made available to the users of a distribution in central repositories, which may contain many thousands of packages, each specially built and maintained for the distribution.

A distribution may maintain several different repositories for different stages of the software development life cycle. For example, there will usually be a testing repository, which contains packages that have just been built and are intended for use by brave souls who are looking for bugs before the packages are released for general distribution. A distribution will often have a development repository where work-in-progress packages destined for inclusion in the distribution's next major release are kept.

A distribution may also have related third-party repositories. These are often needed to supply software that, for legal reasons such as patents or Digital Rights Management (DRM) anticircumvention issues, cannot be included with the distribution. The thirdparty repositories operate in countries where software patents and anticircumvention laws do not apply. These repositories are usually wholly independent of the distribution they support, and to use them one must know about them and manually include them in the configuration files for the package management system.

Package management system consist of two types of tools: low-level tools that handle task such as installing and removing package files, and high level tools that perform metadata searching and dependency resolutions.

Distributions	Low-level tools	High-Level tools
Debian style	dpkg	apt-get, aptitude
Red-hat style	rpm	yum

By using the high-level tools to search repository metadata, one can locate a package based on its name or description,

Style	Commands
Debian	apt-get update apt-cache search search-string
Red-hat	yum search search-string

High-level tools permit a package to be downloaded from a repository and installed with full dependency resolution.

Style	Commands
--------------	-----------------

Debian	apt-get update apt-get install <i>package_name</i>
Red-hat	yum install <i>package_name</i>

If the package has been downloaded from a source than a repository, it can be installed directly using a low-level tool, tough without dependency resolution.

Style	Command
Debian	dpkg --install <i>package_file</i>
Red hat	rpm -i <i>package_file</i>

Package can be uninstalled using either the high-level or low-level tools,

Style	Command
Debian	apt-get remove <i>package_name</i>
Red hat	yum erase <i>package_name</i>

Package management task is keeping the system up to date with the latest packages. The high-level tools can perform this task in one single step.

Style	Command
Debian	apt-get update; apt-get upgrade
Red hat	yum update

If an update package version has been downloaded from a non-repository source, it can be installed replacing the previous version.

```
$ apt-get update
Hit:1 http://archive.ubuntu.com/ubuntu focal InRelease
Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [107 kB]
Get:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease [111 kB]
Get:4 http://archive.ubuntu.com/ubuntu focal-backports InRelease [98.3
kB]
Get:5 http://security.ubuntu.com/ubuntu focal-security/main amd64
Packages [324 kB]
Get:6 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages
[588 kB]
Get:7 http://security.ubuntu.com/ubuntu focal-security/main Translation-
en [75.5 kB]
Get:8 http://security.ubuntu.com/ubuntu focal-security/main amd64 c-n-f
Metadata [5000 B]
Get:9 http://security.ubuntu.com/ubuntu focal-security/restricted amd64
Packages [59.2 kB]
Get:10 http://security.ubuntu.com/ubuntu focal-security/universe amd64
Packages [506 kB]
Get:11 http://archive.ubuntu.com/ubuntu focal-updates/main Translation-en
[150 kB]
Get:12 http://security.ubuntu.com/ubuntu focal-security/universe
Translation-en [62.8 kB]
Get:13 http://security.ubuntu.com/ubuntu focal-security/universe amd64 c-
n-f Metadata [8484 B]
Get:14 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64
Packages [1252 B]
Get:15 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 c-n-f
Metadata [10.3 kB]
Get:16 http://archive.ubuntu.com/ubuntu focal-updates/restricted amd64
Packages [67.1 kB]
Get:17 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64
Packages [666 kB]
Get:18 http://archive.ubuntu.com/ubuntu focal-updates/universe
Translation-en [124 kB]
Get:19 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64 c-n-
f Metadata [12.0 kB]
Get:20 http://archive.ubuntu.com/ubuntu focal-updates/multiverse amd64
Packages [15.1 kB]
Get:21 http://archive.ubuntu.com/ubuntu focal-backports/universe amd64
Packages [4012 B]
Fetched 2995 kB in 11s (265 kB/s)
Reading package lists... Done
```

Style	Command
Debian	<code>dpkg --install package_file</code>
Red Hat	<code>rpm -u package_file</code>

3.5 Basic file editing

In this chapter, we will introduce the vi text editor, one of the core programs in the Unix tradition. We won't become masters in this chapter, but when we are done, we will know how to using vi. To start vi, we simply enter the following command;

```
$ vi
```

A screen like this should be appear;

```
VIM - Vi IMproved

        version 8.1.2269
        by Bram Moolenaar et al.

        Modified by team+vim@tracker.debian.org
        Vim is open source and freely distributable


        Help poor children in Uganda!

type  :help iccf<Enter>          for information


        type  :q<Enter>          to exit
type  :help<Enter>  or  <F1>    for on-line help
type  :help version8<Enter>    for version info
```

:q

:q!

if all goes well, we will get a screen like this;

```
$ rm -f countries.txt
$ vi countries.txt
```

The leading tilde characters (~) indicate that no text exists on that line. This shows that we have an empty file. The second most important thing to learn about vi is that vi is a modal editor. When vi starts up, it begins in command mode. In this mode, almost every key is a command.

In order to add some text to the file, we must first enter insert mode. To do this just press the **I** key. Afterward we should see the following at the bottom of the screen.

```
--INSERT --
```

Now we can enter some text, for the example countries name as the file names is countries.txt

after finish enter text in the file, to exit insert mode and return to command mode, press the Esc key.

Indonesia
Malaysia
Singapore

Filipina
Brunei

To save the change we just made to our file, we must enter an *ex command* while in command mode. This easily done by pressing the `:` key. To write the modified file follow the colon with a `w` then enter.

:W

The file will be written to the hard drive, and we should get a confirmation message at the bottom of the screen;

"countries.txt" [New] 6L, 48C written

While it is in command mode, vi offers a large number of movement commands, some of which it shares with less

Key	Cursor moves
L or right arrow	Right one character
H or left arrow	Left one character
J or down arrow	Down one line
K or up arrow	Up one line
0 (zero)	To beginning current line
Shift-6 (^)	To the first non-white character on the current line
Shift-4 (\$)	To the end of the current line
w	To the beginning of the next word punctuation character
Shift-w (W)	To the beginning of the next word, ignoring punctuation character
B	To the beginning of the previous word or punctuation character
Shift b (B)	To the beginning of the previous word, ignoring punctuation characters
CTRL-F or PAGE DOWN	Down one page
CTRL-B or PAGE UP	Up one page
number-SHIFT-G	To line number (for example, 1 G moves to the first line of the file)
SHIFT-G (G)	To the last line of the file

Most editing consists of a few basic operations such as inserting text, deleting text, and moving text around by cutting and pasting. vi, of course, supports all of these operations in its own unique way. This will come in handy as we try out some of the basic editing commands.

Appending text

Vi/vim provides a command to append text, the sensibly named a command. If we move the cursor to the end of the line and type a, the cursor will move past the end of the line, and vi will enter insert mode. This will allow us to add some more text

3.6 Searching text using regular expression

Searching text within the line can be done using f command, the f command search the current line and moves the cursor to the next instance of the specific character. For example, type the command fa on the command mode of the vim countries.txt file, the command would move the cursor to the next occurrence of the character a within the current line.

To move the cursor to searching the next occurrence of a word/phrase in the entire file, the /command can be used. Practice the command within the exercise file. Next, type /the word or phrase to be search for, followed by the enter key.

```
Indonesia maritime continent
Malaysia
Filiphina
[REDACTED]
Brunai
~
~
~
/Singapore
```

~
~
~

The cursor will move to line contain Singapore word (line 4). Try tp type other words or phrase to practice the searching command.

The program will be use to work with regular expression is grep, the name grep is derived from the phrase global regular expression print. grep searches text files for the occurrence of a

specified regular expression and outputs any line containing a match to standard output. Commonly we used grep with fixed string;

```
$ ls /usr/bin | grep zip
bunzip2
bzip2
bzip2recover
gpg-zip
gunzip
gzip
zipdetails
```

This will list all the files in the /usr/bin directory whose names contain the substring zip.

The grep program accepts options and arguments this way:

`grep [options] regex [file...]`

where *regex* is regular expression

Below is list of commonly used grep option.

Option	Description
-i	Ignore case. Do not distinguish between upper- and lowercase characters. May also be specified --ignore-case.
-v	Invert match. Normally, grep prints lines that contain a match. This option causes grep to print every line that does not contain a match. May also be specified --invert-match.
-c	Print the number of matches (or non-matches if the -v option is also specified) instead of the lines themselves. May also be specified --count.
-l	Print the name of each file that contains a match instead of the lines themselves. May also be specified --files-with-matches.
-L	Like the -l option, but print only the names of files that do not contain matches. May also be specified --files-without-match.
-n	Prefix each matching line with the number of the line within the file. May also be specified --line-number.
-h	For multifile searches, suppress the output of filenames. May also be specified --no-filename.

Below is example of createing some text file to search using grep option:

```
$ ls /bin > examplefile-bin.txt
$ ls /usr/bin > examplefile-usr-bin.txt
$ ls /sbin > examplefile-sbin.txt
$ ls /usr/sbin > examplefile-usr-sbin.txt
$ ls examplefile*.txt
examplefile-bin.txt examplefile-sbin.txt examplefile-usr-bin.txt examplefile-usr-sbin.txt
```

we can perform a simple search of our file list like example below:

```
$ grep bzip examplefile*.txt
examplefile-bin.txt:bzip2
examplefile-bin.txt:bzip2recover
examplefile-usr-bin.txt:bzip2
examplefile-usr-bin.txt:bzip2recover
```

grep searches all of the listed files for the string bzip and finds four matches in the file examplefile-bin.txt and examplefile-usr-bin.txt. If we were interested in only the files that contained matches rather than the matches themselves, we could specify the -l option:

```
$ grep -L bzip examplefile*.txt
```

```
examplefile-sbin.txt
examplefile-usr-sbin.txt
```

Conversely, if we wanted to see a list of only the files that did not contain a match, we could do this:

```
$ grep -l bzip examplefile*.txt
examplefile-bin.txt
examplefile-usr-bin.txt
```

3.6 Manage file permission and ownership

Before we dive into file ownership and permission in linux, we need to know file types in Linux. Regular file, directory, and symbolic link are the most common types that we are working with in this training. File types that are used in linux system:

- Regular file (-)
- Directory / folder (d)
- Symbolic link (l)
- Character device file (c)
- Block device file (b)
- Local socket file (s)
- Named pipe (p)

Linux is a multiuser system and allow us to access the same linux machine concurrently. Linux divides authorization into two levels: Ownership and Permission.

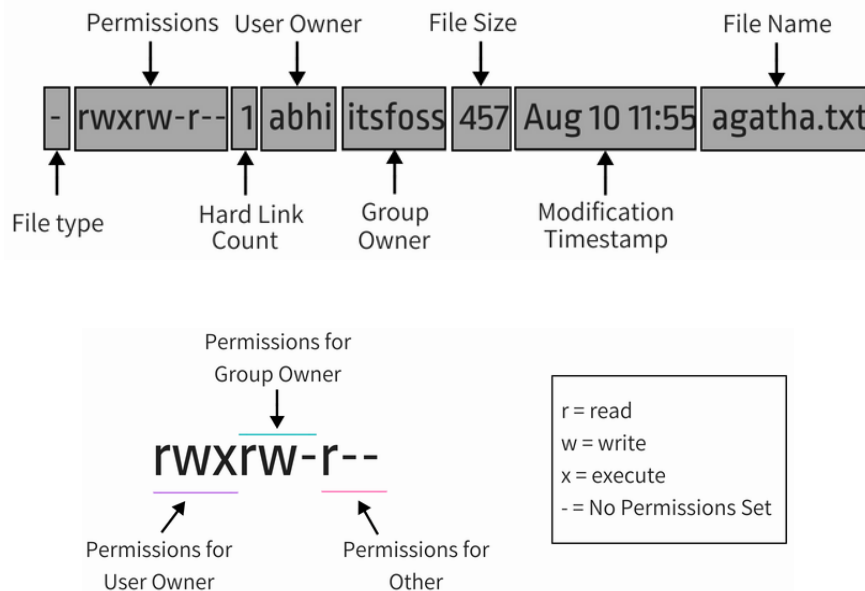
Ownership of Linux files: every file and directory on linux system is assigned three types of owner, given below.

- User (u)
A user is the owner of the file. By default, the person who created a file becomes its owner. Hence, a user is also sometimes called an owner.
- Group (g)
A user- group can contain multiple users. All users belonging to a group will have the same access permissions to the file. Suppose you have a project where a number of people require access to a file. Instead of manually assigning permissions to each user, you could add all users to a group, and assign group permission to file such that only this group members and no one else can read or modify the files.
- Other (o)
Any other user who has access to a file. This person has neither created the file, nor he belongs to a usergroup who could own the file. Practically, it means everybody else. Hence, when you set the permission for others, it is also referred as set permissions for the world.

Permission of Linux files: there are three permissions defined for all three owners discussed above.

- Read (r)
This permission give you the authority to open and read a file. Read permission on a directory gives you the ability to lists its content.
- Write (w)
The write permission gives you the authority to modify the contents of a file. The write permission on a directory gives you the authority to add, remove and rename files stored in the directory. Consider a scenario where you have to write permission on file but do not have write permission on the directory where the file is stored. You will be able to modify the file contents. But you will not be able to rename, move or remove the file from the directory.
- Execute (x)

The write permission gives you the authority to modify the contents of a file. The write permission on a directory gives you the authority to add, remove and rename files stored in the directory. Consider a scenario where you have to write permission on file but do not have write permission on the directory where the file is stored. You will be able to modify the file contents. But you will not be able to rename, move or remove the file from the directory.



3.6.1 Change file permission and ownership

To change file and directory permissions, use the command `chmod` (change mode). The owner of a file can change the permissions for user (u), group (g), or others (o) by adding (+) or subtracting (-) the read, write, and execute permissions. There are two basic ways of using `chmod` to change file permissions: The symbolic method and the absolute / octal form.

Symbolic method is probably easiest way is the relative (or symbolic) method, which lets you specify permissions with single letter abbreviations. A `chmod` command using this method consists of at least three parts from the following lists:

Access class	Operator	Access Type
u (user)	+ (add access)	r (read)
g (group)	- (remove access)	w (write)
o (other)	= (set exact access)	x (execute)
a (all: u, g, and o)		

Example:

```
$ chmod a+x agatha.txt
```

agatha.txt can be executed by all users

```
$ chmod go-rw agatha.txt
```

agatha.txt can be read and written only by user owner

```
$ chmod o-w mydir/
```

removes write permission for other user to mydir directory

```
$ chmod -R o+w mydir/
```

allows write permission for other user to mydir directory and all subdirectories

Absolute / octal method, The other way to use the chmod command is the absolute form, in which you specify a set of three numbers that together determine all the access classes and types. Rather than being able to change only particular attributes, you must specify the entire state of the file's permissions.

The three numbers are specified in the order: user (or owner), group, and other. Each number is the sum of values that specify read, write, and execute access:

4 : read (r)
2 : write (w)
1 : execute (x)

Example:

```
$ chmod 751 agatha.txt
```

grants read, write, and execute permissions to yourself (4+2+1=7), read and execute permissions to users in your group (4+0+1=5), and only execute permission to others (0+0+1=1)

```
$ chmod 700 agatha.txt
```

grants read, write, and execute permissions on the current directory to yourself only

777 anyone can do anything (read, write, or execute)
755 you can do anything; others can only read and execute
711 you can do anything; others can only execute
644 you can read and write; others can only read

3.6.2 Change file ownership

To change the ownership of a file, you can use the command chown. You may easily guess that chown stands for change owner. You can change the user owner of a file in the following manner:

```
chown <NEW_USERNAME> <FILENAME>
```

change the username

```
chown <NEW_USERNAME>:<NEW_USERGROUP> <FILENAME>
```

change the usergroup

```
chown :<NEW_USERGROUP> <FILENAME>
```

```
chgrp <NEW_USERGROUP> <FILENAME>
```

```
$ sudo chown root:root agatha.txt
```

```
$ ls -l agatha.txt
```

```
-rw-rw---- 1 root root 457 Aug 10 11:55 agatha.txt
```

This will change the ownership of the file to root for both user and the group, sudo command is needed because to change ownership to root, this command need superuser right.

3.7 Procces text stream using filters

there are commands to view file content, each command has unique way to view file content.

cat
more
less

shows all file content

head

shows the beginning of file (first 10 lines)

Example:

`head -n5 FILE`
shows the first of 5 lines

tail

shows the end of the file (last 10 lines)

Example:

`tail -n5 FILE`

shows the last 5 lines

`tail -f FILE`

shows the updated last line, it is commonly used to show updated log file

`cut -d DELIMITER -f FIELD FILE`

shows selected field in csv or tabular file

Example:

`cut -d: -f2 FILE`

shows second field with ":" as delimiter

`cut -d, -f3 FILE`

shows third field with "," as delimiter

paste

this command merges lines of files and writes lines consisting of the sequentially corresponding lines from each FILE, separated by TABs, to standard output.

Example: merging two files cities.txt and countries.txt

`$ cat cities.txt`

Indonesia

Malaysia

Phillipines

Thailand

Brunei

`$ cat countries.txt`

Jakarta

Kuala Lumpur

Manila

Bangkok

Bandar Sri Begawan

`$ paste countries.txt cities.txt`

Indonesia Jakarta

Malaysia Kuala Lumpur

Phillipines Manila

Thailand Bangkok

Brunei Bandar Sri Begawan

`$ paste -d: countries.txt cities.txt`

Indonesia:Jakarta

Malaysia:Kuala Lumpur

Phillipines:Manila

Thailand:Bangkok

Brunei:Bandar Sri Begawan

sort

sorts lines of text files.

Example:

`$ sort countries.txt`

Brunei

```
Indonesia
Malaysia
Phillipines
Thailand
$ sort -u FILE
```

sort lines of the text file and only show the unique values

wc

Print newline, word, and byte counts for each FILE, and a total line if more than one FILE is specified. A word is a non-zero-length sequence of characters delimited by white space.

Example:

```
$ wc cities.txt
4 8 54 cities.txt
```

wc command shows that cities.txt file has 4 lines, 8 words, 54 characters

3.8 Use stream, pipes and redirection

The redirection capabilities built into Linux provide you with a robust set of tools used to make all sorts of tasks easier to accomplish. Whether you're writing complex software or performing file management through the command line, knowing how to manipulate the different I/O streams in your environment will greatly increase your productivity.

3.8.1 Streams

Input and output in the Linux environment is distributed across three streams. These streams are:

- standard input (STDIN), number 0
it provides input from keyboard or mouse to commands
- standard output (STDOUT), number 1
it displays output from commands

Example:

```
$ echo "hello"
hello
```

the hello in the second line is the standard output (STDOUT)

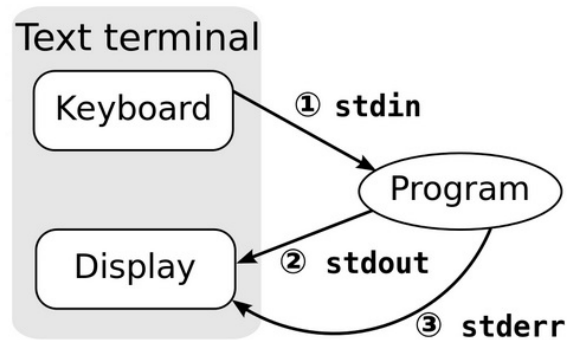
- standard error (STDERR), number 2
it displays error output from commands

Example:

```
$ ls %
ls: cannot access %: No such file or directory
```

Since % is not an existing directory, this will send the following text to standard error (STDERR)

During standard interactions between the user and the terminal, standard input is transmitted through the user's keyboard. Standard output and standard error are displayed on the user's terminal as text. Collectively, the three streams are referred to as the standard streams. The image below will show differences between streams.



3.8.2 Redirection

Linux includes redirection commands for each stream. These commands write standard output to a file. If a non-existent file is targetted (either by a single-bracket or double-bracket command), a new file with that name will be created prior to writing.

Commands with a single bracket overwrite the destination's existing contents.

Overwrite

- > - standard output
- < - standard input
- 2> - standard error

Commands with a double bracket do not overwrite the destination's existing contents.

Append

- >> - standard output
- << - standard input
- 2>> - standard error

Example:

```
$ cat > write_to_me.txt
a
b
c
ctrl-d
```

Here, cat is being used to write to a file, which is created as a result of the loop.

```
$ cat write_to_me.txt
a
b
c
```

View the contents of write_to_me.txt using cat:

```
$ cat > write_to_me.txt
1
2
3
ctrl-d
```

Redirect cat to writetome.txt again, and enter three numbers.

```
$ cat write_to_me.txt
1
2
3
```

The prior contents are no longer there, as the file was overwritten by the single-bracket command.

```
$ cat >> write_to_me.txt
a
b
c
ctrl-d
```

cat redirection with double brackets

```
$ cat write_to_me.txt
1
2
3
a
b
c
```

The file now contains text from both uses of cat, as the second one did not override the first one.

command > file

This pattern redirects the standard output of a command to a file.

```
$ ls ~ > root_dir_contents.txt
```

The command above passes the contents of your system's root directory as standard output, and writes the output to a file named rootdircontents.txt. It will delete any prior contents in the file, as it is a single-bracket command.

command > /dev/null

/dev/null is a special file that is used to trash any data that is redirected to it. It is used to discard standard output that is not needed, and that might otherwise interfere with the functionality of a command or a script. Any output that is sent to /dev/null is discarded.

In the future, you may find the practice of redirecting standard output and standard error to /dev/null when writing shell scripts.

```
$ ls > /dev/null
```

This command discards the standard output stream returned from the command ls by passing it to /dev/null.

command 2> file

This pattern redirects the standard error stream of a command to a file, overwriting existing contents.

```
$ mkdir " " 2> mkdir_log.txt
```

This redirects the error raised by the invalid directory name "", and writes it to log.txt. Note that the error is still sent to the terminal and displayed as text.

command >> file

This pattern redirects the standard output of a command to a file without overwriting the file's existing contents.

```
$ echo Written to a new file > data.txt
$ echo Appended content to an existing file >> data.txt
```


This pair of commands first redirects the text inputted by the user through echo to a new file. It then appends the text received by the second echo command to the existing file, without overwriting its contents.

command 2>> file

The pattern above redirects the standard error stream of a command to a file without overwriting the file's existing contents. This pattern is useful for creating error logs for a program or service, as the log file will not have its previous content wiped each time the file is written to.

```
$ find " 2> stderr_log.txt
$ wc " 2>> stderr_log.txt
```

The above command redirects the error message caused by an invalid find argument to a file named stderr_log.txt. It then appends the error message caused by an invalid wc argument to the same file.

3.8.3 Pipes

Pipes are used to redirect a stream from one program to another. When a program's standard output is sent to another through a pipe, the first program's data, which is received by the second program, will not be displayed on the terminal. Only the filtered data returned by the second program will be displayed. The Linux pipe is represented by a vertical bar.

```
* | *
```

Example:

```
$ ls | less
```

This takes the output of ls, which displays the contents of your current directory, and pipes it to the less program. less displays the data sent to it one line at a time.

Filters

Filters are commands that alter piped redirection and output. Note that filter commands are also standard Linux commands that can be used without pipes.

find - Find returns files with filenames that match the argument passed to find.

grep - Grep returns text that matches the string pattern passed to grep.

tee - Tee redirects standard input to both standard output and one or more files.

tr - tr finds-and-replaces one string with another.

wc - wc counts characters, lines, and words.

Examples

Now that you have been introduced to redirection, piping, and basic filters, let's look at some basic redirection patterns and examples.

command | command

Redirects the standard output from the first command to the standard input of the second command.

```
$ find /var lib | grep rpm
```

This command searches through /var and its subfolders for filenames and extensions that match the string deb, and returns the file paths for the files, with the matching portion in each path highlighted in red.

command | tee file

This pattern (which includes the tee command) redirects the standard output of the command to a file and overwrites its contents. Then, it displays the redirected output in the terminal. It creates a new file if the file does not already exist.

In the context of this pattern, tee is typically used to view a program's output while simultaneously saving it to a file.

```
$ wc /etc/magic | tee magic_count.txt
```

This pipes the counts for characters, lines, and words in the magic file (used by the Linux shell to determine file types) to the tee command, which then splits wc's output in two directions, and sends it to the terminal display and the magic_count.txt file. For the tee command, imagine the letter T. The bottom part of the letter is the initial data, and the top part is the data being split in two different directions (standard output and the terminal).

command | command | command >> file

This pattern predirects the standard output of the first command and filters it through the next two commands. It then appends the final result to a file.

```
$ ls ~ | grep *tar | tr e E >> ls_log.txt
```

This begins by running ls in your root directory (~) and piping the result to the grep command. In this case, grep returns a list of files containing tar in their filename or extension.

The results from grep are then piped to tr, which replaces occurrences of the letter e with E, since e is being passed as the first argument (the string to search for), and E is passed as the second argument (the string that replaces any matches for the first argument). This final result is then appended to the file ls_log.txt, which is created if it does not already exist).

3.9 Perform basic file management

ls

we will continue using ls command in more practical use case.

Example:

```
$ ls -a
```

shows all files and directories including hidden files. Hidden file / directory in linux has . (dot) prefix, i.e .ssh/ directory, .bashrc and .bash_logout file.

```
$ ls -ltr *.txt
```

shows only file which as .txt extension in list with details, sorted by time in reverse

```
$ ls /usr/bin
```

shows /usr/bin directory contents

touch

it makes empty file(s)

Example:

```
$ touch temp_20201001.txt temp_20201002.txt
```

Create two empty files with filename temp_20201001.txt temp_20201002.txt

cp

copy files and directories

Example:

```
$ cp temp_20201001.txt Documents/
```

Copy file temp_20201001.txt to Documents/ directory

```
$ cp *.txt Downloads/
```

Copy all file with .txt extension to Downloads directory

```
$ cp -r Downloads/ download_backup/
```

Copy Downloads directory to download_backup directory

mv

this command move or rename files or directories

Example:

```
$ mv temp_20201001.txt temperature_20201001.txt
```

Rename file temp_20201001.txt to temperature_20201001.txt

```
$ mv temperature_20201002.txt Downloads/
```

Move file temperature_20201001.txt to Downloads directory

```
$ mv temperature_20201002.txt \ Downloads/temperature_20201001.txt
```

Move file temperature_20201001.txt to Downloads directory and rename it to temperature_20201001.txt

```
$ mv Documents/ Archived/
```

Rename Documents directory to Archived

rm

delete files or directories

Example:

```
$ rm temperature_20201001.txt
```

Delete file temperature_20201001.txt

```
$ rm -fr download_backup/
```

Delete download_backup directory

tar

this command is utility for archiving, allows us to create a compressed file (like zip and rar in windows) and reserve the original files or directories

Example:

```
$ tar zcfv download_archived.tar.gz download_backup/
```

Create archive file with .tar.extension of download_backup directory and name it as download_archived.tar.gz

```
$ tar xzfv download_archived.tar.gz
```

Extracts download_backup.tar.gz file

```
$ tar tf download_backup.tar.gz
```

Show file list in download_backup.tar.gz, without extract it

gzip and gunzip

this tools are for archiving and extracting .gz file (not .tar.gz). the main difference wuth tar command is these tools delete the original files or directories.

Example:

```
$ gzip temperature_20201002.txt
```

Archives temperature_20201002.txt to temperature_20201002.txt.gz as default name and delete the original file

```
$ gunzip temperature_20201002.txt.gz
```

Extracts temperature_20201002.txt.gz and delete the original file

CHAPTER IV Linux Processes Monitoring and Task Scheduling

Success Indicators	The participants are expected can manage the processes, working with linux linked files, locate searched file, understand shell environment, make simple shell scripts to be scheduled, and demonstrate compilation from small package application source code.
---------------------------	---

4.1 Monitor and Kill Processes

uptime

shows how long the machine has been running, uptime gives a one line display of the following information. The current time, how long the system has been running, how many users are currently logged on, and the system load averages for the past 1, 5, and 15 minutes.

ps

report a snapshot of the current processes. ps displays information about a selection of the active processes.

Example :

```
$ ps aux
```

top

displays Linux processes. The top program provides a dynamic real-time view of a running system. It can display system summary information as well as a list of processes or threads currently being managed by the Linux kernel. The types of system summary information shown and the types, order and size of information displayed for processes are all user configurable and that configuration can be made persistent across restarts.

free

Display amount of free and used memory in the system. free displays the total amount of free and used physical and swap memory in the system, as well as the buffers and caches used by the kernel.

watch

executes a program periodically, showing output fullscreen. watch runs command repeatedly, displaying its output and errors (the first screenfull). This allows you to watch the program output change over time. By default, command is run every 2 seconds and watch will run until interrupted.

Example:

```
$ watch ls wrfout*
```

executes ls wrfout* command every 2 second, this command is very useful for watching wrfout files during running the WRF.

pgrep

pgrep looks through the currently running processes and lists the process IDs which match the selection criteria to stdout. All the criteria have to match.

kill

send a signal to a process. The default signal for kill is TERM. Use -l or -L to list available signals. Particularly useful signals include HUP, INT, KILL, STOP, CONT, and 0. Alternate signals may be specified in three ways: -9, -SIGKILL or -KILL. Negative PID values may be used to choose whole process groups; see the PGID column in ps command output. A PID of -1 is special; it indicates all processes except the kill process itself and init.

Example:

```
$ kill 2913
```

This command kill a process with process ID (PID) 2913. PID can be found with ps or pgrep command

```
$ kill -9 2913
```

The option -9 will force kill PID 2913

killall

it will kill processes by specified name. killall sends a signal to all processes running any of the specified commands, the option -9 also works with this command.

Example:

```
$ killall -9 firefox
```

This command forces kill all firefox processes.

4.2 Create Links

ln -s TARGET LINK_NAME

soft link is a special kind of file that points to another file, much like a shortcut. it is useful when we need to access a file or directory with desired filename without copy or rename the original file and we can still do common file operation same like the original one. if the link was deleted, the original file still exist. if the original file was deleted, the link still exist but became a broken link.

ln TARGET LINK_NAME

4.3 Find System Files

whereis

it returns the location of binary, source, and manual page of commands in the linux system.

Example:

```
$ whereis ls
ls: /usr/bin/ls /usr/share/man/man1/ls.1.gz
```

which

this command returns the location of the binary of commands in the linux system.

Example:

```
$ which ifconfig
/sbin/ifconfig

$ which ln
/usr/bin/ln
```

find FOLDER [OPTIONS]

searchs file with specific filter (i.e name, modified or creation time, user permission)

Example:

find files which has .jpg extension in curent folder

```
$ find . -name '*.jpg'
```

```
./image_1.jpg
./image_2.jpg
./image_5.jpg
./image_6.jpg
```

find files which has .log extension in /var/log folder, and show their first 10 lines

```
$ find /var/log -name '*.log' -exec head {}
/var/log/tuned/tuned.log
/var/log/audit/audit.log
/var/log/anaconda/anaconda.log
/var/log/boot.log
/var/log/yum.log
```

locate KEYWORD

This command searches file based on indexed file, it's only available in RedHat-based linux distribution. if the locate failed, it means the file index database is updated yet, to update it, type sudo updatedb

4.5 Automate Tasks by Scheduling Jobs

Crontab uses crond as the daemon. Cron is a Linux based utility for scheduling time-based jobs that run automatically at a set time, date or after a specific interval. You can automate various repetitive administrative tasks (e.g. database backups, email reminders, etc) using cron jobs. The beauty of cron job lies on accuracy. Since the overall process of running the tasks is automated, you can use this utility to execute mission critical instructions without ever forgetting.

Example of job definition:

```
..... minute (0 - 59)
| ..... hour (0 - 23)
| | ..... day of month (1 - 31)
| | | ..... month (1 - 12) OR jan,feb,mar,apr ...
| | | | ..... day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
| | | |
m h dom mon dow usercommand
* * * * *
```

Example:

```
0 * * * * /folder/youscript.sh
execute yourscrip.sh every hour on 00 minute
*/10 * * * * /folder/youscript.sh
every 10 minutes
0 7-9 1 * * /folder/youscript.sh
every first day of the month on 07:00 08:00 09:00
0 0,2,5,8 * * * /folder/youscript.sh
every 00:00 02:00 05:00 08:00
```

4.6 Make and install programs from source

```
./configure
make
make install
```

Despite using package management for application installation, it's possible to install from the source code. WRF installation uses this method because WRF offers many options of library and compiler. here is an example for make and install a zlib library that required by WRF.

download zlib source package

```
$ wget -c \
https://www2.mmm.ucar.edu/wrf/OnLineTutorial/compile_tutorial/tar_files/zlib-1.2.7.tar.gz
```

extracting the source package

```
$ tar xzfv zlib-1.2.7.tar.gz
```

change directory to extracted package

```
$ cd zlib-1.2.7/
$ ./configure --prefix=`pwd`
```

Configure the software. The `./configure` script is responsible for getting ready to build the software on your specific system. It makes sure all of the dependencies for the rest of the build and install process are available, and finds out whatever it needs to know to use those dependencies. Unix programs are often written in C, so we'll usually need a C compiler to build them. In these cases the `./configure` script will establish that your system does indeed have a C compiler, and find out what it's called and where to find it.

```
$ make
```

Build the software. Once `configure` has done its job, we can invoke `make` to build the software. This runs a series of tasks defined in a Makefile to build the finished program from its source code. The tarball you download usually doesn't include a finished Makefile. Instead it comes with a template called `Makefile.in` and the `configure` script produces a customised Makefile specific to your system.

```
$ make install
```

Install the software. Now that the software is built and ready to run, the files can be copied to their final destinations. The `make install` command will copy the built program, and its libraries and documentation, to the correct locations. This usually means that the program's binary will be copied to a directory on your `PATH`, the program's manual page will be copied to a directory on your `MANPATH`, and any other files it depends on will be safely stored in the appropriate place. Since the `install` step is also defined in the Makefile, where the software is installed can change based on options passed to the `configure` script, or things the `configure` script discovered about your system. Depending on where the software is being installed, you might need escalated permissions for this step so you can copy files to system directories. Using `sudo` will often do the trick.

```
$ ls -ltr
```

check the compiled source. should you see ``bin`, `lib`, `include`` folders

CHAPTER V Linux Shell Programming

5.1 Working with Shell Scripting

There are two major types unix shell :

- Bourne shell, it has \$ character as the default prompt and has subcategories ; Bourne shell (sh), korn shell (ksh), Bourne Again shell (bash), POSIX shell (sh), Z shell (zsh)
- C shell, this shell has %character as the default prompt and has subcategories; C shell (csh) and TENEX/TOPS C shell (tcsh)

Bash shell is the most popular, mostly we will use this shell in this course and some parts is C shell.

5.1.1 Shell Environment

env

it show all environment variables at current session

export

it assigns an environment variable with a value

Example:

```
$ export COUNTRY=indonesi
$ echo $COUNTRY
indonesia
```

```
$ export NETCDF=/usr
$ echo $NETCDF
```

alias

it helps us for creating alias command, it is very useful to simplify long command to be shorted.

Example:

```
$ ls -ltr
$ alias lt='ls -ltr'
$ lt
```

source

The source command reads and executes commands from the file specified as its argument in the current shell environment. It is useful to load functions, variables, and configuration files into shell scripts.

Example:

Open file editor and write as below, and save it as compiler.sh

```
export CC=gcc
export CXX=g++
export FC=gfortran
export F77=gfortran
export FFLAGS=-m64
```

```
$ source compiler.sh
$ echo $CC
```

.bashrc

This file is located in home folder (~) and will be loaded when the session is started. Mostly contains environment variables, functions and aliases definition as defaults for terminal.

5.1.2 Write Simple Script

Your first shell script, open text editor and write as follows:

```
#!/bin/bash

# my first shell script

echo "hello world"
```

save the file as hello.sh and back to terminal.

```
$ bash hello.sh
hello world

$ chmod +x hello.sh
$ ./hello.sh
hello world
```

for

this shell keyword command allows us to do looping of commands in shell, there are other commands that can do looping, i.e while and until.

Example:

```
$ for PARM in temp rh wspeed wdir
do
    echo $PARM
done
```

it's an example loop from a list, will have an output, as below:

```
temp
rh
wspeed
wdir

$ for FILETXT in `ls *.txt`
do
    echo $FILETXT
done
```

loop with STDIN from `ls *.txt` command, will have an output as below:

```
temp_20201001.txt
temp_20201002.txt
temp_20201003.txt
```

5.1.3 Variables

Assigning variable

```
DAY=sunday
DAY="sunday"
```

Displaying variable

```
echo $DAY          => sunday
echo ${DAY}        => sunday
```

Slicing string

```
echo ${DAY:0:2}    => su
echo ${DAY::2}     => su
echo ${DAY::-1}    => sunda
echo ${DAY:2}      => nday
```

```
echo ${DAY:3:2} => da
```

5.1.4 Shell Arguments

It is very common in linux command to have arguments, with arguments will allow us to have user input without changing the script.

\$ command ARGUMENT1 ARGUMENT2

To get the value of these arguments, we need special character \$N. The N is the Nth argument value. Let's make a script to show the arguments of the command.

```
#!/bin/bash

echo "filename = $0"
echo "first argument = $1"
echo "second argument = $2"
echo "third argument = $3"
echo "number of arguments = $#"
```

Execute the script

```
$ chmod +x shell.arg.sh
$ ./shell.arg.sh argument1 argument_number_2 argument_three
filename = ./shell.arg.sh
first argument = argument1
second argument = argument_number_2
third argument = argument_three
number of arguments = 3
```

5.2 Shell Programming

Shell programming allows us to perform many linux commands combined into one script. All of the command that have been introduced in previous chapters can be run sequentially to do some particular task. Shell script will help our workflow in running WRF use case, i.e downloading GFS files needs wget command to do downloading and loop it with for keyword. Post-processing the WRF output utilizes grads command to produce forecast images.

5.3 Create simple script for downloading GFS

WRF can be run by downscaling GFS global model. GFS is downloaded from NOMADS NCEP NOAA website

<https://nomads.ncep.noaa.gov/pub/data/nccf/com/gfs/prod> and available in 00, 06, 12, 18 model run. Usually it's ready to download 4 hours 40 minutes after each model run, i.e 00 model run is available at 04.40 UTC and so on.

```
folder /pub/data/nccf/com/gfs/prod/gfs.{DATE}/{RUN}/
file   gfs.t{RUN}z.pgrb2.0p25.f{FR}
```

DATE	date with YYYYMMDD format
RUN	initial run 00 06 12 18
FR	forecast range 000 until 120 in hourly

wget is a tool for downloading file, we only need to do looping wget command

open an text editor, write as below, save it as download-gfs.sh

```
#!/bin/bash

DATE=$1
HOUR=$2

for FR in `seq -f "%03g" 0 3 24`
do
    wget -c \
    https://nomads.ncep.noaa.gov/pub/data/nccf/com/gfs/prod/gfs.${DATE}/${RUN}/gfs.t${RUN}z.p
    grb2.0p25.f${FR}
done

$ chmod +x download-gfs.sh
$ ./download-gfs.sh 20201005 00
```

NOAA also provides GFS filtered download; it allows us to download selected parameters, layers, and domain which saves the bandwidth and download time. We can access the filtered GFS with this link

https://nomads.ncep.noaa.gov/cgi-bin/filter_gfs_0p25.pl and will guide us to select parameters, layers, and domain that we need. Below is the example of GFS grib filter URL

https://nomads.ncep.noaa.gov/cgi-bin/filter_gfs_0p25.pl?file=gfs.t00z.pgrb2.0p25.f000&all_lev=on&all_var=on&subregion=n=&leftlon=90&rightlon=150&toplat=20&bottomlat=-20&dir=%2Fgfs.20201023%2F00

That URL will download GFS 0.25 Indonesia domain 20N - 20S, 90W – 150W, all_lev=on means all levels, all_var=on means all parameters on 2020-10-23 00H model run at forecast range 000H. To simplify the URL, we choose all variables and all levels.

We will make a script to download filtered GFS with specific domain and model run as user input. The script will download next until 24 hours forecast range in 3 hourly.

```
$ ./download.gfs.filter.sh DATE RUN
```

```
#!/bin/bash

DATE=$1
HOUR=$2
LLON=90
RLON=150
TLAT=20
BLAT=-20

for FR in `seq -f "%03g" 0 3 24`
do
    wget -c \
    "https://nomads.ncep.noaa.gov/cgi-bin/filter_gfs_0p25.pl?
    file=gfs.t${HOUR}z.pgrb2.0p25.f${FR}&
    all_lev=on&all_var=on&subregion=&
    leftlon=${LLON}&rightlon=${RLON}&toplat=${TLAT}&bottomlat=${BLAT}&dir=gfs.${DATE}/${HOUR}z.pgrb2.0p25.f${FR}"
done
```

```

leftlon=90&\
rightlon=150&\
toplat=20&\
bottomlat=-20&\
dir=%2Fgfs.${DATE}%2F${HOUR}"\
-O gfs.${DATE}.${HOUR}.${FR}.grib2
done

```

```

$ chmod +x ./download.gfs.filter.sh
$ ./download.gfs.filter.sh 20201023 00

```

Exploring GRIB2 parameters list

Mainly we use `wgrib2` command to explore GRIB2. In this case, we are going to see parameters in GFS 0.25 grib file and uses the `gfs.grib2` file in the example folder.

Display list of parameters of GFS

```

$ wgrib2 gfs.grib2
1:0:d=2020102300:CLMR:1 hybrid level:anl:
2:24430:d=2020102300:ICMR:1 hybrid level:anl:
3:24609:d=2020102300:RWMR:1 hybrid level:anl:
4:63589:d=2020102300:SNMR:1 hybrid level:anl:
5:63768:d=2020102300:GRLE:1 hybrid level:anl:
...
520:24648841:d=2020102300:PRMSL:mean sea level:anl:
521:24716922:d=2020102300:5WAVH:500 mb:anl:
522:24780153:d=2020102300:LANDN:surface:anl:

```

So many lines printed, to make easier to read, we can pipe (|) the output to other command (i.e less, more)

```

$ wgrib2 gfs.grib2 | less
$ wgrib2 gfs.grib2 | more

```

Count parameters based on lines printed, thanks to `wc` command, it will display 522 lines which means the GRIB2 file has 522 parameters

```

$ wgrib2 gfs.grib2 | wc -l
522

```

We also can display the unique parameters inside of the GRIB2 file with `cut`, `wc` and `sort` commands. Utilize `cut` command displays only parameter field (located in 4th field) which separated by colon (:) and continue piping with `sort` command to sort the parameters and eliminated the duplicated parameters

```

$ wgrib2 gfs.grib2 | cut -d: -f4 | sort -u
4LFTX
5WAVH
ABSV
APTMP
...
VVEL
VWSH
WEASD
WILT

```

Bellow is common command pattern

Pattern	Description
*	Match zero or more characters
?	Match any single characters
[...]	Match of any characters in a set
?(patterns)	Match zero or one occurances of the patterns (extglob)
*(patterns)	Match zero or more occurances of the patterns (extglob)
+(patterns)	Match one or more occurances of the patterns (extglob)
@(patterns)	Match one occurances of the patterns (extglob)
!(patterns)	Match anything that doesn't match one of the patterns (extglob)

Example

```
daniel@DESKTOP-A5BD6CM:~$ ls
a.out countries.txt examplefile-sbin.txt examplefile-usr-sbin.txt cities.txt
examplefile-bin.txt examplefile-usr-bin.txt hellow.c

daniel@DESKTOP-A5BD6CM:~$ ls *.txt
cities.txt countries.txt examplefile-bin.txt examplefile-sbin.txt
examplefile-usr-bin.txt examplefile-usr-sbin.txt

daniel@DESKTOP-A5BD6CM:~$ ls ?.out
a.out

daniel@DESKTOP-A5BD6CM:~$ ls [ab]*
a.out
```

```
daniel@DESKTOP-A5BD6CM:~$ shopt -s extglob # turn on the extended globing

daniel@DESKTOP-A5BD6CM:~$ ls ?(*.txt|*.out)
a.out      countries.txt      examplefile-sbin.txt      examplefile-usr-
sbin.txt
cities.txt examplefile-bin.txt examplefile-usr-bin.txt

daniel@DESKTOP-A5BD6CM:~$ ls !(*.txt|*.out) # not a .txt or a .out file
hellow.c
```

Below is Regularly used commands

Command	Description
cd	To changing directory
ls	list directory contents and information about the files, by default sorted alphabetically.
df	To see the current amount of free space on the disk drive
pwd	shows the current/working directory, telling where you are currently located in filesystem
mkdir	Make a new directory
mkdir grib	Make grib directory
rm dir	Delete empty directory
chmod	Change file and directory permission
chown	Change the ownership of a file
cat	To view file content / show all file content
more	

less	
head	Shows the beginning of file (first 10 lines)
tail	Shows the end of the file (last 10 lines)
cut	shows selected field in csv or tabular file
paste	merges lines of files and writes lines consisting of the sequentially corresponding lines from each FILE, separated by TABs, to standard output.
sort	sorts lines of text files.
wc	Print newline, word, and byte counts for each FILE, and a total line if more than one FILE is specified.
find	This command searches through /var and its subfolders for filenames and extensions that match the string deb, and returns the file paths for the files, with the matching portion in each path highlighted in red.
touch	It make(s) empty file
cp	copy files and directories
mv	move or rename files or directories
rm	delete files or directories
tar	utility for archiving, allows us to create a compressed file (like zip and rar in windows) and reserve the original files or directories
gzip	archiving and extracting .gz file
gunzip	
uptime	shows how long the machine has been running
ps	report a snapshot of the current processes
top	displays Linux processes
free	Display amount of free and used memory in the system
watch	executes a program periodically, showing output fullscreen
pgrep	looks through the currently running processes and lists the process IDs which match the selection criteria to stdout
kill	send a signal to kill a process
ln	a special kind of file that points to another file, much like a shortcut
whereis	returns the location of binary, source, and manual page of commands in the linux system.
which	returns the location of the binary of commands in the linux system
find	searchs file with specific filter
locate	searchs file based on indexed file
env	show all environment variables at current session
export	assigns an environment variable with a value
alias	creating alias command
source	The source command reads and executes commands from the file specified as its argument in the current shell environment
echo	save the file and back to terminal.
for	to do looping of commands in shell
./configure	responsible for getting ready to build the software on your specific system.
make	Build the software
make	Install the software
install	
ls -ltr	check the compiled source

CHAPTER VI Summary and Quiz

6.1 Summary

This linux course just a beginning to understand NWP and also could be useful for linux beginner migrating from Windows. Inevitably, mostly NWP model runs on linux operation system because linux is easy to scale. NWP is the best way to simulate and predict atmospheric processes, so that NHMS could provide weather services to reduce the risk of weather phenomena. Despite the physics and thermodynamics understanding, linux skill is the first needed to transform the complicated physical equations that run on the NWP.

In real life, Linux also could be operating system alternative from Windows. More linux distributions has been developed with better graphical interface and easier to use by linux beginner. Familiarizing with using linux in daily basis is great way learn linux so that improve the technical skill that needed to run NWP.

6.2 Quiz

1. What command clears the contents of your terminal display?
2. What is the command to delete a file?
3. What do you type in to move to the parent directory?
4. What command is used to change directories?
5. What is the default directory path for system log files?
6. What command is used to get the ip address of all interfaces on a server?
7. What command and parameter (or switch) will force a program to quit (even one running in the background)?
8. What command is used to change ownership of a file?
9. What command is used to copy a file?
10. What command(s) shows you disk partitions and percentage of disk space used?
11. What command shows you what directory you are in?
12. What command creates an empty directory?
13. What command displays your current username?
14. What command shows you CPU and memory utilization for running processes?
15. What command allows you to open and view a file one page at a time?
16. Which command(s) show users that are logged in?
17. What command is used to change a file name?
18. What is the command to switch to the root user account?
19. What command is used to change the permissions of a file?
20. What command is used to display your previous commands?

REFERENCES

1. <https://opensource.com/resources/linux>
2. <https://www.redhat.com/en/topics/open-source/what-is-open-source>
3. <https://www.howtogeek.com/196060/beginner-geek-how-to-create-and-use-virtual-machines/>
4. <https://www.guru99.com/file-permissions.html>
5. <https://linuxhandbook.com/linux-file-permissions/>
6. <https://kb.iu.edu/d/abdb>
7. <https://borosan.gitbook.io/lpic1-exam-guide/1034-use-streams-pipes-and-redirects>
8. <https://www.digitalocean.com/community/tutorials/an-introduction-to-linux-i-o-redirection>
9. <https://www.baeldung.com/linux/pipes-redirection>
10. <https://blog.usejournal.com/what-is-the-difference-between-a-hard-link-and-a-symbolic-link-8c0493041b62>
11. <https://www.tutorialspoint.com/unix/unix-what-is-shell.htm>
12. https://medium.com/@Alibaba_Cloud/how-to-automate-and-schedule-tasks-with-crontab-on-ubuntu-16-04-54c091922d2c
13. Linux Manual Pages
14. <https://devhints.io/bash>
15. <https://medium.com/sysf/bash-scripting-everything-you-need-to-know-about-bash-shell-programming-cd08595f2fba>
16. https://linuxhint.com/30_bash_script_examples/
17. <http://matt.might.net/articles/bash-by-example/>
18. <https://www.shellscript.sh/>
19. https://en.wikibooks.org/wiki/Bash_Shell_Scripting
20. <https://swcarpentry.github.io/shell-novice/>