Software Development Life Cycle

Lecture 4: Linux and Shell Script

DR. YOUNG SAENG PARK

2021/2022

WMG
THE UNIVERSITY OF WARWICK

Content

- Introduction of Linux
- Linux Command Line
- Users and Groups
- Services and Processes
- Package Management
- Shell Script

Linux and Shell Script

Introduction of Linux

Linux

- **Linux** is a family of **open-source Unix-like operating systems** based on Linux kernel.
- Linux was first released in 1991 by **Linux Torvalds** who also created the distributed revision control system **Git**.
- Linux is typically packaged in a **Linux distribution** which includes the **Linux kernel** and **supporting system software libraries**, **many** of which are provide by the **GNU Project**.
- Linux was originally **developed for personal computers** based on the Intel x86 architecture, but has since been **ported to more platforms**.
- Because of the dominance of the **Linux-based Android** on smartphones, Linux is **one of most popular general-purpose operating systems**.
- Linux also **runs on embedded systems** including routers, automation controls, smart home controls, automobiles, etc.

Linux

Source: Logic Supply

Linux – Everywhere

- **Linux** has found **everywhere** such as Smartphone, Network applications, TVs, Servers, Games, etc.
- Linux is **working in every areas** that Microsoft Windows is used, and many more.
- Some **noteworthy products** using Linux are:
  - Android phones – manufactured by Samsung, HTC, Huawei, LG, etc.
  - Samsung Galaxy Watch
  - Linksys Routers
  - Samsung TVs (Most models)
  - Sony BRAVIA TVs
  - Netgear NAS (Network-attached storage)
  - LG Hisense Washing Machines

WMG
THE UNIVERSITY OF WARWICK

Linux – Reasons to Use

Runs on Many Hardware Platforms

- Proprietary Unix operating systems typically on run on their hardware from their company. For example, HP-UX only runs on HP servers, AIX only runs on IBM servers.

Small Footprint

- The small footprint on Linux allows it run on older hardware or on embedded systems.

Stable, Reliable and Secure

- This makes it a great choice for servers that need to continuously run without down time.

Great for Servers

- Linux has traditionally been used for server applications. Linux can be used to host websites, act as file servers, and run as database servers.

Free/Open Source Software

- Linux is free, not only the source code but also possible to run Linux on any hardware without having to pay a licensing fee in many cases.

Linux Distribution

- **Linux distribution** is a **collection of applications, packages, managements, and features** that run on top of the **Linux kernel**.
- **Linux users** usually obtain their operating system by **downloading one of the Linux distributions**.
- Linux distributions differ in serval ways, and three of most important are: **Purpose**, **Configuration and packaging**, and **Support model**.
- **Most of the included software** in a Linux distribution is **free and open-source software** made available both as **complied binaries** and in **source code form**.
- **Almost one thousand Linux distributions exist** and some of well-known Linux distributions are: **Debian**, **Ubuntu**, **Fedora**, **CentOS**, **openSUSE**, etc

Source: The Ultimate Linux Newbie Guide

Popular Linux Distribution (Distro)

- **Arch:** it is popular amongst developers. It is an independently developed system. It is designed for users who go for a do-it-yourself approach.

- **CentOS:** it is one of the most used Linux Distro for enterprise and web servers. It is a free enterprise class OS and is based heavily on Red Hat enterprise Distro.

- **Debian**: it is a stable and popular non-commercial Linux Distro. It is widely used ad a desktop Linux Distro and is used-oriented.

- **Fedora:** it is supported by the Fedora project, an endeavour by Red Hat. It is popular among desktop users. Its versions are known for their short life cycle.

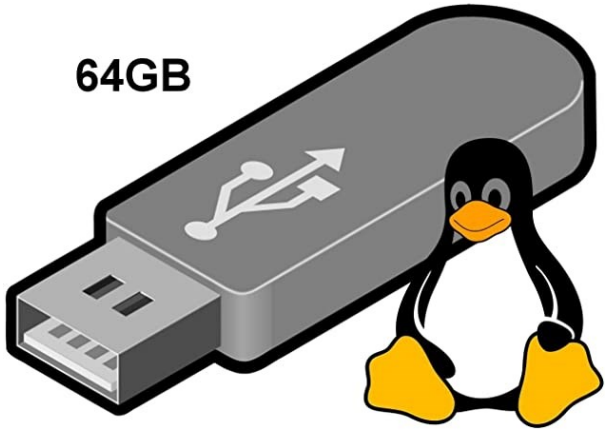
- **OpenSUSE:** it is an easy to use and a good alternative to MS windows. It can be easily set up and can also run on small computers with obsolete configurations.

- **Ubuntu**: This is the third most popular desktop operating system after Microsoft Windows and Apple Mac OS. It is based on the Debian Linux Distro and it is known as its desktop environment.

Linux Installation using USB stick

This is one of the easiest methods of installing any Linux Distribution on a computer.

- **STEP 1:** Download the .iso or the OS files on a local computer from Linux Distribution website.
- **STEP 2:** Download free software (e.g. Universal USB installer) to make a bootable USB stick.
- **STEP 3:** Select .iso file downloaded from STEP 1 and select a USB to create a bootable USB stick.
- **STEP 4:** Make sure to boot from USB via the boot option in the BIOS setup.
- **STEP 5:** Put the bootable USB stick into a local computer and restart it. Then, follow the installation procedure.



Source: Amazon.co.uk

Linux Installation using Virtual Machine

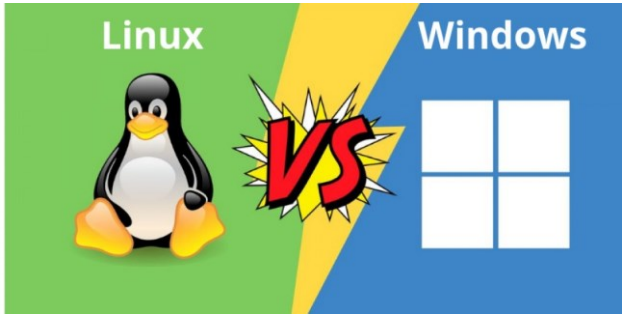
The virtual installation offers the freedom of running Linux on an existing OS already installed on a computer.

- **STEP 1:** Download an install Virtual Machine software like Virtual Box or VMware.
- **STEP 2:** Download the .iso on a local computer from Linux Distribution website.
- **STEP 3:** Create new Virtual Machine and select which OS version to be installed.
- **STEP 4:** Select the Virtual Machine environment such size of memory, size of disk space, etc.
- **STEP 5:** Make sure where .iso file is located in a computer.
- **STEP 6:** Follow the installation procedure which might be different based on the Virtual Machine software.

ORACLE VM VirtualBox

Linux vs Windows

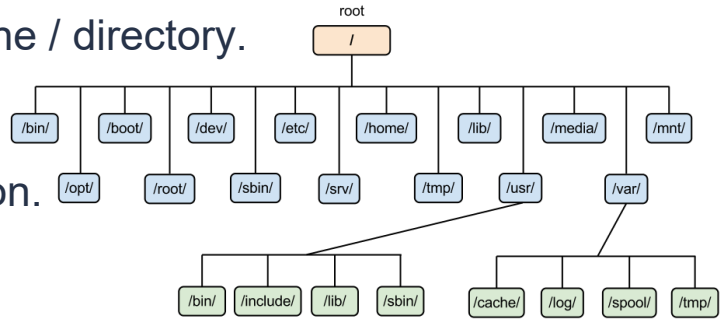
- **Linux** is an **open source operating systems** whereas **Windows** is a **commercial OS** belonging to Microsoft.
- Linux runs **faster** even with older hardware but Windows are **slower** compared to Linux.
- **Administrator user** has all administrative privileges of computers in Windows whereas **Root user** is the super user and has all administrative privileges in Linux.
- Windows file naming convention is **not case sensitive** but Linux is **case sensitive**. For example, test, Test and TEST are 3 different files in Linux
- In Windows, **C:\Users\account** is default home directory whereas for every users **/home/username** directory is creased as home directory.
- **Hard drives and printers in Windows** are considered **as devices** but **Linux** considers them **as files**



Source: Hackr.io

Linux File System Hierarchy

- **/ as the root directory has everything** located under the / directory.
- **/bin** contains **essential user binaries**.
- **/sbin** contains essential binaries for system administration.
- **/boot** contains the **files needed to boot** the system.
- **/etc** contains **configuration files** which can generally be edited by hand in a text editor.
- **/home** contains a **home folder for each user**. e.g. for **yspark** account, the home folder is **/home/yspark**.
- **/lib** contains **libraries** needed by the essential binaries in the **/bin** and **/sbin** folder.
- **/var** contains files to which the **system writes data** during operation.
- **/usr** contains applications and files used by users.
- **/tmp** stores **temporary files** which are generally **deleted after restart**.



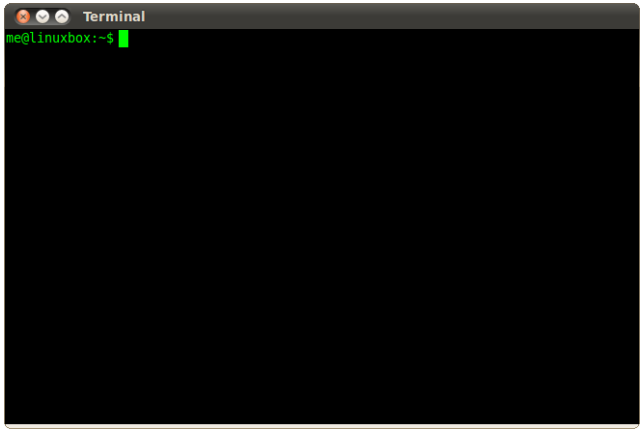
Linux and Shell Script

Linux Command Line

WMG
THE UNIVERSITY OF WARWICK

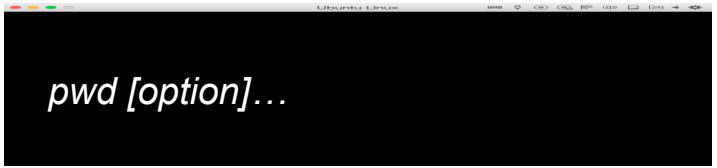
Command Line Interface (CLI)

- **A command line interface (CLI)** processes **commands** to a computer program **in the form of lines of text**.
- Operating systems implement a **shell for interactive access** to operating system **functions or services**.
- The CLI gives a **better understanding of how Linux works** behind the scenes.
- There is a **terminal application** given for interactive access and also a **text editor** such vim, nano to create a shell commands script file.
- Although many users rely upon graphical user interface, the command line interface is **very useful in a case no graphical user interface**.
- It may **be difficult for a new user** to become familiar with **all the commands and options** available.



Linux Command – pwd

- **The pwd** command is one of the most frequently used Linux utilities.
- The pwd stands for "**print working directory**".
- It **checks the full path of where you are** currently located within the Linux File System.
- It is a very **useful command** if you **get lost where you are** currently working on.
- The option ***'-L'*** prints the value of $PWD from environment which contains the current working directory.
- The option **'-P'** prints the **physical directory**, **without any symbolic links**.
- By default, the pwd behavers **as if '-L' were specified**.



Linux Command – ls

- **The ls** command is one of the basic Linux commands that allows **to list files or directories**.
- The ls **accepts some flags** which are **additional information to change how files or directories** are listed in a terminal.
- **‘ls [directory path]’** command is to list the contents of another directory.
- **‘ls ..’** is to list the contents of the **parent directory** one level above.
- **‘ls ~’** is to list the contents in the user’s home directory
- **‘ls –l’** is to list **the contents of the directory in a table format** with columns including: content permissions, number of links to the content, owner of the content, group owner of the content, size of the content in bytes, last modified date/time, file or directory name.

ls [flags] [directory]

```
→  ~ ls
Applications            Library                 Public
Desktop                 Movies                  Virtual Machines.localized
Documents               Music                   new-moon.itermcolors
Downloads               Pictures                ~
→  ~
```

```
→  ~ ls ~
Applications            Library                 Public
Desktop                 Movies                  Virtual Machines.localized
Documents               Music                   new-moon.itermcolors
Downloads               Pictures                ~
→  ~
```

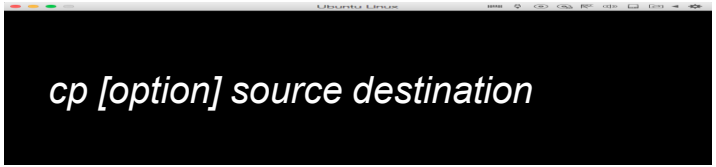
```
→  ~ ls -l
total 24
drwx------@   5 bolajiayodeji  staff   160 Jun 16 03:35 Applications
drwx------+   8 bolajiayodeji  staff   256 Jul 11 06:56 Desktop
drwx------@  10 bolajiayodeji  staff   320 Jul 22 22:31 Documents
drwx------@ 168 bolajiayodeji  staff  5376 Aug 17 05:48 Downloads
drwx------@  70 bolajiayodeji  staff  2240 Jul 20 20:52 Library
drwx------+  11 bolajiayodeji  staff   352 Jul 23 10:29 Movies
drwx------+   6 bolajiayodeji  staff   192 Jun 17 03:25 Music
drwx------+   7 bolajiayodeji  staff   224 Jul 12 15:16 Pictures
drwxr-xr-x+   4 bolajiayodeji  staff   128 Jun 16 01:28 Public
drwxr-xr-x    4 bolajiayodeji  staff   128 Jul 27 13:49 Virtual Machines.localized
-rwxr--r--@   1 bolajiayodeji  staff  9261 Jun 16 03:16 new-moon.itermcolors
drwxr-xr-x    3 bolajiayodeji  staff    96 Jun 17 03:38 ~
→  ~
```

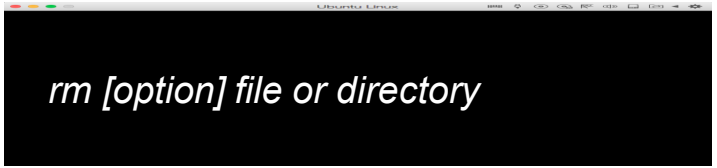
Linux Command – cd

- **The cd** command is **used to change the current working directory**.
- The cd is one of the most basic and **frequently used commands** when working on the Linux terminal.
- **Like pwd**, the cd has **options '-L' and '-P**' which are working similarly and '-L' option is specified as default.
- When specifying a directory to change, it accepts either **absolute or relative path names**.
- The **absolute** or full path starts from **the system root, '/'.**
- **'cd ~'** is to change into home director.
- The **current working directory** is represented by a **single dot (.)**. **Two dots (..)** represents the **parent directory** or the directory above the current one.

Linux Command – cp, mv, rm

- **The cp command** is used **to copy a file or a group of files or a directory**.
- The cp requires **at least two filenames** in its arguments: **source and destination**.
- There are **many options** of cp command, -i (interactive), -b (backup), -f (force), -r (coping directory structure), etc.
- **The mv command** is used **to move one or more files or directories** from one place to another.
- The mv is also used **to rename a file or folder**.
- **The rm command** is used **to remove a file or a group of files or a directory**.
- **'rm –r'** recursively deletes a directory and all its contents, **'rm –f'** forcibly deletes files without asking.





Linux Command – mkdir, rmdir

- **The mkdir command** is used **to create a directory or multiple directories**.
- The mkdir command **must have enough permissions** to create a directory in the parent directory.
- It is possible to create a directory **not only in the current working directory but also at any place using absolute path**.
- **The rmdir command** is used **to delete an empty directory**.
- If the specified directory has **some sub-directories or files in it**, this cannot be removed by rmdir command and so use '-f' option.
- To delete a directory and all of its contents recursively, use **'rm –r'** instead.

Linux Command – Others (cat, touch, grep, sudo)

- **The cat command** is used to **display the contents of a file** on the standard output (stdout).
- Here are **other ways to use the cat command**:
  - **'cat > filename'** creates a new file.
  - **'cat filename1 filename2 > filename3'** joins two files (1 and 2) and stores the output of them in a new file (3).
- **The touch command** allows to **create a blank new file** through the Linux command line.
- **The grep command** is to **search through all the text** in a given file**.**
  - For example, **'grep blue notepad.txt'** will search for the word blud in the notepad file. Lines that contain the searched word will be displayed fully.
- **The sudo command** is short for "SuperUser Do" and enables to perform tasks that require **administrative or root permissions**.
  - It is not advisable to use the sudo command and be careful because it might be easy to make an error if there is something wrong.

Linux Command – Others (locate, find, df, du)

- **The locate command** is used to **locate a file** like the search command in Windows and the option, **'–i',** argument along with this command will **make it case insensitive**.
  - To search for a file that contain two or more words, **use an asterisk (*)**.
  - For example, **'locate –i student*name'** command will search for any file that contains the word 'student' and 'note', whether it is uppercase or lowercase.
- **The find command** similar to the locate command is to **search for files and directories**. The difference is that the find command is a more aggressive search tool than the locate.
  - For example, **'find /home/ -name notes.txt'** command will search for a file called **'notes.txt'** within the home directory and its subdirectories.
  - To find files in **the current directory** use, **'find . –name notes.txt'**.
- **The df command** is to get a **report on the system's disk space usage**, shown in percentage and KBs. If you want to see the report in megabytes, type **'df –m'**.
- **The du command** is to **check how much space a file or a directory takes** with disk block numbers instead of the size format. Use **'–h'** option for bytes, kilobytes, etc.

Linux Command – Others (head, tail, diff, tar)

- **The head command** is used to **view the first lines of any text file**. By default, it will show the first ten lines but it is manageable by an option.
  - For example, if you only want to show the first five lines, use **'head –n 5 filename.txt'**.
- **The tail command** is similar function to the head command. The tail command will display the last ten lines of a text file.
  - For example, to show the last five lines, use **'tail –n 5 filename.txt'**.
- **The diff command compares the contents of two files line by line**. The simplest form of this command is **'diff file1.ext file2.ext'**.
- **The tar command** is the most used command to **archive multiple files into a tarball** which is a common Linux file format **similar to zip format**.
- The tar command is **complex with a long list of options** such as adding new files into an existing archive, listing the content of an archive, extracting the content from an archive, and many more.

Linux Command – Others (chmod, chown, kill)

- **The chmod command** is used to change the read, write, and execute permissions of files and directories.
- **The chown command** enables to change or transfer the ownership of a file to the specific username.
  - For example, **'chown user1 filename.ext'** will make user1 as the owner of the 'filename.ext'.
- **The kill command** terminates a process manually. It **sends a certain signal to the process** and instructs the process to terminate itself.
  - There is a total of sixty-four signals but people usually only use two signals.
  - **SIGTERM (15)** – requests a process to stop running and gives some time to save all of its progress.
  - **SIGKILL (9)** – forces a process to stop immediately.
  - It typically uses **ps command** first to find the process identification number **(PID)** of a process to kill.
- **The ps command**, short for Process Status, is a command to **display or view information related to the processes** running in a Linux system.

Linux Command – Others (ping, top, history, man, wget)

- **The ping command** is used to check connectivity status to a server.
  - For example, **'ping google.com'** checks the connection to Google and also measure the response time.
- **The top command** displays a **list of running processes** and **how much CPU each process uses**.
- It is very useful to **monitor system resource usage**, especially knowing which process needs to be terminated in a case that it consumes too many resources.
- **The history command** keeps a **list of all the other commands that have been run**, then allows to **replay or reuse those commands** instead of retyping them.
- **The man command** is to display the instruction of how to use each Linux command.
  - For example, **'man ps'** show the manual instruction of the ps command.
- **The wget command** allows to **download files from the internet.**
  - For example, **'wet https://xxx.yyy.zzz/file.zip'** download **'file.zip'** file from **'xxx.yyy.zzz'**

Linux and Shell Script

Users and Groups

Add a User

- **Linux** was designed to **allow more than one user** to have access to the system at the same time.
- To be able to create a user account, it needs to be **logged in as root or user with sudo privileges**.
- It is possible to **create a new user** from the **command line or** through the **GUI**.
- Typically, there are **two commands** to create a new user account: **useradd** and **adduser**.
- **useradd** is **a low-level utility** for adding users, while the **adduser** is **a friendly interactive frontend** to useradd written in Perl.
- To create a new user account named **‘username’** using the adduser command, run

  ‘sudo adduser username’.

Source: BigBoxCode

Delete a User

- To be able to delete a user account, it needs to be **logged in as root or user with sudo privileges**.
- It is possible to **delete a new user** from the **command line or** through the **GUI**.
- Similar to create a user account, there are **two commands** to create a new user account: **userdel** and **deluser**
- It recommends the **deluser** command as it is **more friendly** than **the low-level userdel**.
- To delete the user, without removing the user file, run

  ‘sudo deluser username’

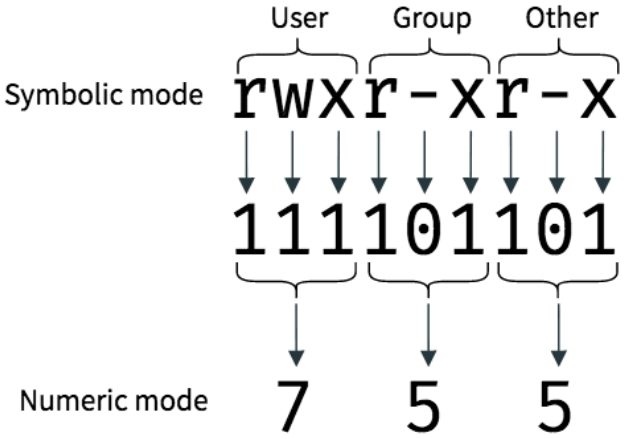
- If you want to **delete the user and its home directory and mail spool**, use the **‘–remove-home’** option.

  ‘sudo deluser –remove-home username’

Source: BigBoxCode

Ownership of Linux

- Each file and directory on Linux system **is assigned 3 types of ownership**: **User**, **Group**, and **Other**.
- **User**
  - A user is the owner of the file or directory. A user is also sometimes called an owner.
  - By default, the person who created a file becomes its owner.
- **Group (User Group)**
  - A group can contain multiple users.
  - All users belonging to a group will have the same Linux group permissions access to the file.
- **Other**
  - This person has neither created the file, nor he belongs to a user group who could own the file.
  - It is referred as set permissions for everybody else.



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

Example of Changing Ownership

Change the **owner** of a file

```
# ls -lart tmpfile
-rw-r--r-- 1 himanshu family 0 2012-05-22 20:03 tmpfile

# chown root tmpfile

# ls -l tmpfile
-rw-r--r-- 1 root family 0 2012-05-22 20:03 tmpfile
```

Change the **group** of a file

```
# ls -l tmpfile
-rw-r--r-- 1 himanshu family 0 2012-05-22 20:03 tmpfile

# chown :friends tmpfile

# ls -l tmpfile
-rw-r--r-- 1 himanshu friends 0 2012-05-22 20:03 tmpfile
```

Change **both owner** and **group**

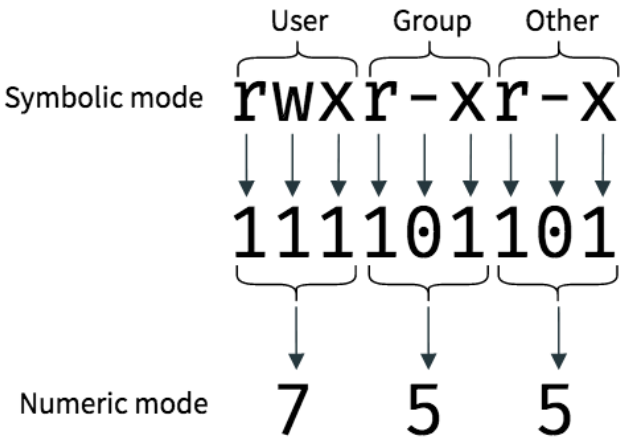
```
# ls -l tmpfile
-rw-r--r-- 1 root family 0 2012-05-22 20:03 tmpfile

# chown himanshu:friends tmpfile

# ls -l tmpfile
-rw-r--r-- 1 himanshu friends 0 2012-05-22 20:03 tmpfile
```

Permission Types

- Every file and directory in Linux system has **3 permissions defined for all the 3 owners: Read, Write, and Execute**.
- **Read (r)**
  - This permission gives the authority to open and read a file.
  - Read permission on a directory gives the ability to list its contents.
- **Write (w)**
  - The Write permission gives the authority to modify a file.
  - The Write permission on a directory gives the authority to add, remove and rename files stored in the directory.
- **Execute (x)**
  - Unlike Windows (extension ‘exe’), an executable program in Linux **requires the execute permission set**.
  - If the execute permission is not set, it might be able to see/modify the program code in a case of having Read and Write permissions.



Example of Permission

```
-rw-rw-r--
drw-r--r--
```

- The first **'-'** implies **a file** and **'d'** implies **a directory**.
- The first part of `'rw-'` means that the owner can **read and write** the file but can **not execute** the file.
- The second part of `'rw-'` means that the group can **read and write** the file but can **not execute** the file.
- The third part of `'r--'` is for the other which can **only read** the file.

0	**No Permission**	`---`
1	**Execute**	`--x`
2	**Write**	`-w-`
3	**Execute + Write**	`-wx`
4	**Read**	`r--`
5	**Read + Execute**	`r-x`
6	**Read + Write**	`rw-`
7	**Read + Write + Execute**	`rwx`

Linux and Shell Script

Services and Processes

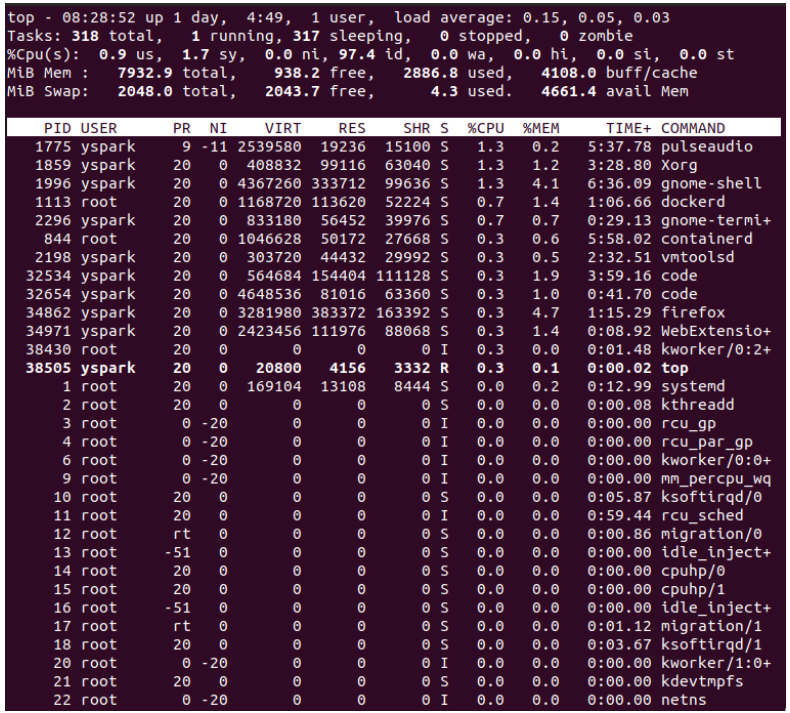
Services and Processes

- A lot of **background activities and server applications** runs as services.
- **Services, also called daemons**, run many of the key functions on a host.
- **Each service** or demon is **one or more processes** running on a host.
- These processes have names, e.g., **sshd** (the Secure Shell demon), **httpd** (the Apache web server), **mysqld** (the MySQL database server), etc.
- Some of these processes may be **running by default** on a host together **with a number of other processes** that perform a variety of system and application functions.
- Most daemon process usually **have a name ending in “d”** for daemon.

```
yspark@yspark-virtual-machine:~/Workspace/Script$ ps -A
    PID TTY          TIME CMD
      1 ?        00:00:12 systemd
      2 ?        00:00:00 kthreadd
      3 ?        00:00:00 rcu_gp
      4 ?        00:00:00 rcu_par_gp
      6 ?        00:00:00 kworker/0:0H-kblockd
      9 ?        00:00:00 mm_percpu_wq
     10 ?        00:00:05 ksoftirqd/0
     11 ?        00:00:57 rcu_sched
     12 ?        00:00:00 migration/0
     13 ?        00:00:00 idle_inject/0
     14 ?        00:00:00 cpuhp/0
     15 ?        00:00:00 cpuhp/1
     16 ?        00:00:00 idle_inject/1
     17 ?        00:00:01 migration/1
     18 ?        00:00:03 ksoftirqd/1
     20 ?        00:00:00 kworker/1:0H-kblockd
     21 ?        00:00:00 kdevtmpfs
     22 ?        00:00:00 netns
     23 ?        00:00:00 rcu_tasks_kthre
     24 ?        00:00:00 rcu_tasks_rude_
     25 ?        00:00:00 rcu_tasks_trace
     26 ?        00:00:00 kauditd
     28 ?        00:00:00 khungtaskd
     29 ?        00:00:00 oom_reaper
     30 ?        00:00:00 writeback
     31 ?        00:00:00 kcompactd0
     32 ?        00:00:00 ksmd
     33 ?        00:00:01 khugepaged
     80 ?        00:00:00 kintegrityd
     81 ?        00:00:00 kblockd
     82 ?        00:00:00 blkcg_punt_bio
     83 ?        00:00:00 tpm_dev_wq
     84 ?        00:00:00 ata_sff
     85 ?        00:00:00 md
```

Services and Processes – top

- The ***top*** command can show **which processes are running** on a host and **which are consuming** the most CPU and memory.
- The ***top*** command starts an interactive monitoring tools that **updates every few seconds** with the top running processes on a host.
- The top shows a lot of detail including system **uptime**, **system load average**, **CPU usage**, **memory utilisation**, etc.
- Some column headings in the process list are:
  - **PR**: Process priority
  - **NI**: the nice value of the process
  - **VIRT**: Amount of virtual memory used by the process
  - **RES**: Amount of resident memory used by the process
  - **SHR**: Amount of shared memory used by the process
  - **S**: Status of the process



Services and Processes – systemd

- The ***ps* command** with the **-A flag** (for all) **lists all the processes** currently running on a host. e.g., ***ps -A***
- Each process running on a host is listed **in order of its Process ID (PID)** used to control processes.
- The most important process on a host is called **systemd.**
- The **systemd** process is **the base process** on Linux hosts that **spawns all other processes** on a host.
- The **systemd** process always **uses PID 1** and **must be running** for a host to be functional.
- You may find **init** process instead of **systemd** process. The **systemd** is recent addition to mainstream Linux OS's. So, you might see init process on older systems.

systemd

```
yspark@yspark-virtual-machine:~/Workspace/Script$ ps -A
    PID TTY          TIME CMD
      1 ?        00:00:12 systemd
      2 ?        00:00:00 kthreadd
      3 ?        00:00:00 rcu_gp
      4 ?        00:00:00 rcu_par_gp
      6 ?        00:00:00 kworker/0:0H-kblockd
      9 ?        00:00:00 mm_percpu_wq
     10 ?        00:00:05 ksoftirqd/0
     11 ?        00:00:57 rcu_sched
     12 ?        00:00:00 migration/0
     13 ?        00:00:00 idle_inject/0
     14 ?        00:00:00 cpuhp/0
     15 ?        00:00:00 cpuhp/1
     16 ?        00:00:00 idle_inject/1
     17 ?        00:00:01 migration/1
     18 ?        00:00:03 ksoftirqd/1
     20 ?        00:00:00 kworker/1:0H-kblockd
     21 ?        00:00:00 kdevtmpfs
     22 ?        00:00:00 netns
     23 ?        00:00:00 rcu_tasks_kthre
     24 ?        00:00:00 rcu_tasks_rude_
     25 ?        00:00:00 rcu_tasks_trace
     26 ?        00:00:00 kauditd
     28 ?        00:00:00 khungtaskd
     29 ?        00:00:00 oom_reaper
     30 ?        00:00:00 writeback
     31 ?        00:00:00 kcompactd0
     32 ?        00:00:00 ksmd
     33 ?        00:00:01 khugepaged
     80 ?        00:00:00 kintegrityd
     81 ?        00:00:00 kblockd
     82 ?        00:00:00 blkcg_punt_bio
     83 ?        00:00:00 tpm_dev_wq
     84 ?        00:00:00 ata_sff
     85 ?        00:00:00 md
```

Services and Processes – systemctl (1)

- **The systemctl command** is a utility which is responsible for **examining and controlling the systemd system**.
- **[unit…]** is the name(s) of the systemd unit being queried or managed.
- ‘**systemctl**’ command without arguments will display the status of all units that systemd has loaded. This is the same as the ‘systemctl list-units’ command.
- ‘**systemctl list-unit-files**’ command will list all units that systemd has files for and their state:
  - Static – Can not be enabled
  - Enabled – Can be stared (could already be started)
  - Disabled – Can’t be started but can be triggered from another unit.
  - Masked – Can’t be enabled or triggered from another unit.

systemctl [options] Command [unit…]

```
UNIT FILE                                   STATE           VENDOR PRESET
proc-sys-fs-binfmt_misc.automount           static          enabled
-.mount                                     generated       enabled
boot-efi.mount                              generated       enabled
dev-hugepages.mount                         static          enabled
dev-mqueue.mount                            static          enabled
proc-sys-fs-binfmt_misc.mount               disabled        enabled
run-vmblock\x2dfuse.mount                   enabled         enabled
snap-code-70.mount                          enabled         enabled
snap-code-72.mount                          enabled         enabled
snap-core-11420.mount                       enabled         enabled
snap-core-11606.mount                       enabled         enabled
snap-core18-2074.mount                      enabled         enabled
snap-core18-2128.mount                      enabled         enabled
snap-gnome\x2d3\x2d34\x2d1804-66.mount      enabled         enabled
snap-gnome\x2d3\x2d34\x2d1804-72.mount      enabled         enabled
snap-gtk\x2dcommon\x2dthemes-1514.mount     enabled         enabled
snap-gtk\x2dcommon\x2dthemes-1515.mount     enabled         enabled
snap-snap\x2dstore-518.mount                enabled         enabled
snap-snap\x2dstore-547.mount                enabled         enabled
snap-snapd-12704.mount                      enabled         enabled
snap-snapd-12883.mount                      enabled         enabled
sys-fs-fuse-connections.mount               static          enabled
sys-kernel-config.mount                     static          enabled
sys-kernel-debug.mount                      static          enabled
sys-kernel-tracing.mount                    static          enabled
acpid.path                                  enabled         enabled
apport-autoreport.path                      enabled         enabled
cups.path                                   enabled         enabled
systemd-ask-password-console.path           static          enabled
systemd-ask-password-plymouth.path          static          enabled
systemd-ask-password-wall.path              static          enabled
session-2.scope                             transient       enabled
accounts-daemon.service                     enabled         enabled
acpid.service                               disabled        enabled
alsa-restore.service                        static          enabled
alsa-state.service                          static          enabled
alsa-utils.service                          masked          enabled
anacron.service                             enabled         enabled
apparmor.service                            enabled         enabled
```

Services and Processes – systemctl (2)

- **‘systemctl status [unit]’** (or service [unit] status) will **display the detailed status** of a specific unit.
- **‘systemctl start [unit]’** (or service [unit] start) will **start the specific unit**. If this unit has a startup dependency on other units, and they are not running, they will be started as well.
- **‘systemctl stop [unit]’** (or service [unit] stop) will **stop the specific unit**. If other units have shutdown dependencies on this unit, and they are running, they will be stopped as well.
- **‘systemctl restart [unit]’** (or service [unit] restart) will stop then start the specific unit.
- **‘systemctl reload [unit]’** (or service [unit] reload) will request the specified unit to reload its configuration file.

Services and Processes – systemctl (3)

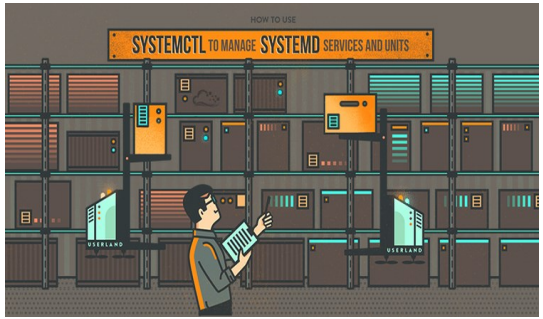
- **‘systemctl roboot’** will **reboot the system** and come up to the default target. This is equivalent to the command **‘shutdown -r’**.
- **‘systemctl enable [unit]’** and **‘systemctl disable [unit]’** will allow the specified unit to be started (enable) or disallow that unit from being started (disable).
- **‘systemctl reenable [unit]’** will disable then enable the specified unit. The unit will not be stopped or started.
- **‘systemctl isolate [target]’** will stop all running units not specified for the target and start non-running units that are specified in for the target.
- **‘systemctl set-default [target]’** will set the specified target to be the default.
- **‘systemctl cat [unit]’** will display the files related to the specified unit.

Services and Processes – systemctl (4)

- There are more commands for **systemctl**:
  - systemctl mask [unit]
  - systemctl unmask [unit]
  - systemctl add-wants [unit]
  - systemctl add-requires [unit]
  - systemctl snapshot [unit]
  - systemctl delete [unit]

Systemd – Configuration Files and Directories

- Systemd uses a number of configuration files to perform its functions. When **running as init (PID = 1)**, the following files are used to define its actions:
  - /etc/systemd/system.conf
  - /etc/systemd/system.conf.d/*.conf
  - /run/systemd/system.conf.d/*.conf
  - /usr/lib/systemd/system.conf.d/*.conf
- Systed uses a number of configuration files to perform its functions. When **running as a user process (PID != 1)**, the following files are used to define its actions:
  - /etc/systemd/user.conf
  - /etc/systemd/user.conf.d/*.conf
  - /run/systemd/user.conf.d/*.conf
  - /usr/lib/systemd/user.conf.d/*.conf

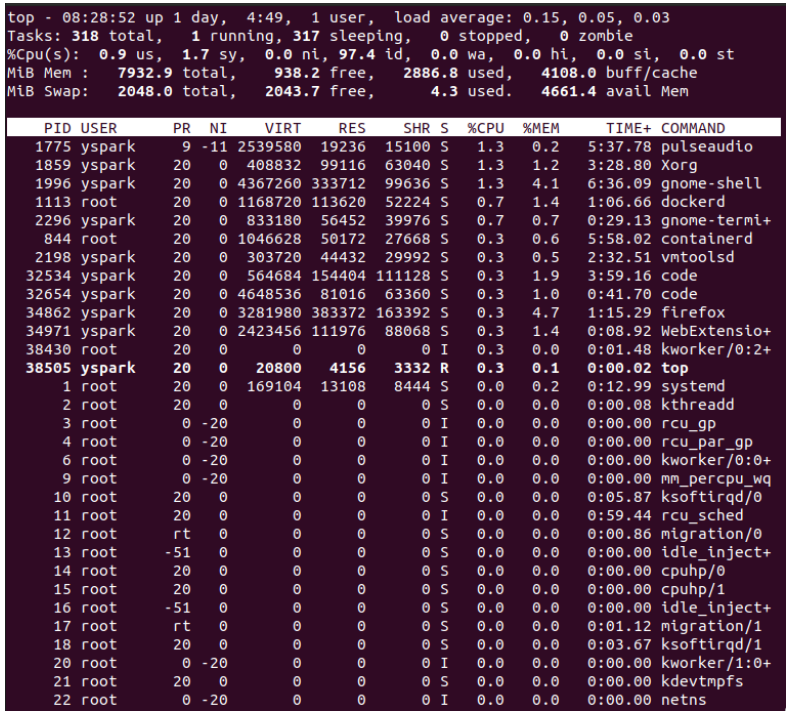


Source: DigitalOcean

- Systemd uses **three directory trees to store configuration files**. These directories, and their uses are as follows:
  - /etc/systemd - Local configuration
  - /run/systemd - Temporary files
  - /usr/lib/systemd - Vendors and Upstream providers

Services and Processes – top

- The ***top*** command can show **which processes are running** on a host and **which are consuming** the most CPU and memory.
- The ***top*** command starts an interactive monitoring tools that **updates every few seconds** with the top running processes on a host.
- The top shows a lot of detail including system **uptime**, **system load average**, **CPU usage**, **memory utilisation**, etc.
- Some column headings in the process list are:
  - **PR**: Process priority
  - **NI**: the nice value of the process
  - **VIRT**: Amount of virtual memory used by the process
  - **RES**: Amount of resident memory used by the process
  - **SHR**: Amount of shared memory used by the process
  - **S**: Status of the process

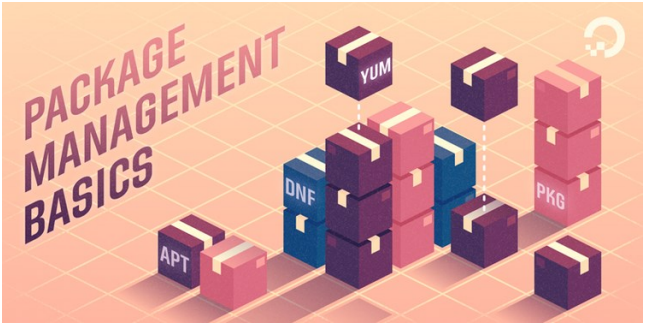


Linux and Shell Script

Package Management

Package Management

- Package management is **a method of installing and maintaining** (which includes updating and probably removing) **software** on a host.
- Different distributions have **different way of packaging their software** by default-prebuilt programs.
- For example, Fedora, Red Hat and CentOS use the **rpm** (Red Hot Package Management) package format, while Ubuntu uses the **deb** (short for Debian, the distribution Ubuntu was originally based on) format.
- Almost all the software that is installed on modern Linux system will **be found on the Internet**. It can be **provided by the distribution vendor through central repositories**.
- Each of these package type uses different tools, such as **dpkg, apt, rpm, yum, zypper**, to manage packages.



Source: DigitalOcean

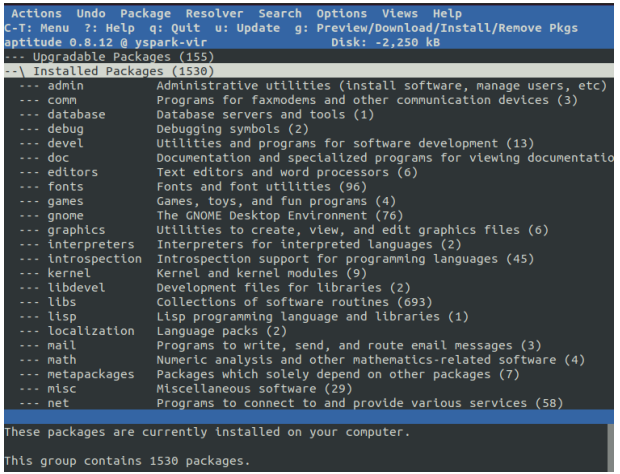
Package Management - Categories

- Although all Linux distributions can contain tens of thousands of packages, broadly speaking these **packages fall into three main categories**.
  - Application packages
  - Library packages
  - Development packages
- Application packages mostly **contain applications**. These applications could range from a simple command-line editor to the whole LibreOffice productivity suite.
- Library packages **contain files that are used by applications and the operating system** to provide additional functionality. For instance, cryptography support is provided by the libssl packages.
- Development packages **contain source code and header files that are required to compile software** from source.

Source: PNGkey

Package Management – Ubuntu Command Line Tools

- The most common way to add software on Ubuntu hosts is to install packages from online software repositories via **command-line tools such as apt, dkpg, aptitude**.
- The **aptitude** is **a text-menu interface**, allowing a user to view the list of packages and to perform management tasks. Actions may be **also performed from the command line**.
- The **dpkg** just **install .deb file**, but will **not install its dependencies** because it does not have those files and it does not have access to repositories to pull the dependencies from.
- The **apt** will **look for the dependencies** and install them. The apt will **automatically check for and get relevant dependencies** to make sure whatever you are trying to install functions correctly.



Package Management – APT

- The apt works with Ubuntu's **Advanced Packaging Tool** (APT) performing such functions as **installation of new software packages**, **upgrade of existing software packages**, **updating of the package list index**, and even **upgrading the entire Ubuntu system**.
- Some example commands of uses for the apt are:
  - Install a package: **sudo apt install nmap**
  - Remove a package: sudo apt remove nmap
  - Update the package index: sudo apt update
  - Upgrade packages: sudo apt upgrade
- You may specify **multiple packages** to be installed or removed, **separated by spaces**.
- Actions of the apt command are logged in the **/var/log/dpkg.log** log file.

Source: Daily Dose of Tech

Linux and Shell Script

Shell Script

Schell Script

- **Shell script** is a computer program **designed to be run by the Unix/Linux shell**, **a command line interpreter**, which cloud be one of the following:
  - The Bourne Shell
  - The C Schell
  - The Korn Shell
  - The GNU Bourn-Again Shell
- Typical operations performed by shell scripts include **file manipulation**, **program execution**, **printing text**, **etc**.
- Shell script is a program to **write a series of commands** for the shell to execute.
- It can combine **lengthy and repetitive sequences of commands** into a single and simple script that **can be stored as a file** and executed anytime.

Source: Fiverr

Schell Script

- A **shell** is a UNIX term for **an interface between a user and an operating system server**.
- Shell provides users with an interface like **accepts human-readable commands** into the system; **executes those commands**; **give the program's output**.
- A shell is **accessed by a terminal** which users type commands.
- When you run the terminal, **the shell issues a command prompt**, where you can type your input, which is then executed when you hit the Enter key.
- The **$ character** is the default prompt in the **Bourne shell**, the **% character** is the default prompt in the **C shell**.
- Generally, you can **customise the command prompt** the environment variable.

Source: Reddit

Write Schell Script

- Shell scripts are **written using text editors** such as vi, nano, visual studio codes, etc.
- The following steps are in creating a shell script:
  1. Create a file using any editor
  2. Name script file with any name and optionally .sh extension (e.g., testshell.sh)
  3. Start the script with #! /bin/sh
  4. Write some code
  5. Save the script file
  6. Execute the script by typing **'sh testshell.sh'**
- **"#!"** is an operator called **shebang** or **hashbang** which directs the script to the interpreter location.
- If we use "#! /bin/sh", the script get directed to the Bourne-shell.

testshell.sh

```
1  #! /bin/sh
2  echo "Hello World"
3
```

```
yspark@yspark-virtual-machine:~/Workspace/Script$ sh testshell.sh
Hello World
yspark@yspark-virtual-machine:~/Workspace/Script$
```

Write Schell Script – Variables (1)

- A **variable** is a character string to which we **assign a value**. The value assigned could be a **number**, **text**, **filename**, **device**, or any other type of data.
- The name of a variable can contain only **letters** (a to z or A to Z), **numbers** (0 to 9) or the **underscore** character (_).
- When you assign a value, **do not leave any space between equal sign** e.g., NAME = “YSPARK” (X), NAME=“YSPARK” (O).
- **By convention**, it is common to have a variable’s name **in UPPERCASE** but it is up to you.
- To access the value stored in a variable**, prefix its name with the dollar sign** e.g., $NAME.
- **To unset a variable**, there is ***unset*** command available.

testshell.sh

```
1  #! /bin/sh
2  NAME="Young Saeng Park"
3  echo "My name is " + $NAME
4  unset NAME
5  echo "My name is " + $NAME
6
```

```
yspark@yspark-virtual-machine:~/Workspace/Script$ sh testshell.sh
My name is  + Young Saeng Park
My name is  +
yspark@yspark-virtual-machine:~/Workspace/Script$
```

Valid

```
_ALI
TOKEN_A
VAR_1
VAR_2
```

Invalid

```
2_VAR
-VARIABLE
VAR1-VAR2
VAR_A!
```

Write Schell Script – Variables (2)

- Shell provides a way to **mark variables as read-only** by using the **readonly** command.
- After a variable is marked read-only, its value **cannot be changed**.
- When a shell is running, **three main types of variables** are present:
  - **Local Variables** – A local variable is a variable that is present **within the current instance of the shell**. It is not available to programs that are started by the shell.
  - **Environment Variables** – An environment variable is **available to any child process of the shell**. Some programs need environment variables in order to function correctly.
  - **Shell Variables** – A shell variable is **a special variable that is set by the shell** and it required by the shell in order to function correctly.

testvariable.sh

```
1  #! /bin/sh
2  NAME="Young Saeng Park"
3  echo "My name is " + $NAME
4  readonly NAME
5  NAME="Superman"
```

```
yspark@yspark-virtual-machine:~/Workspace/Script$ sh testvariable.sh
My name is  + Young Saeng Park
testvariable.sh: 5: NAME: is read only
yspark@yspark-virtual-machine:~/Workspace/Script$
```

Write Schell Script – Array

- A shell supports **an array** which is **containing multiple values** at the same time.
- Instead of creating a new name for each variable that is required, you can use **a single array variable that stores all the other variables**.
- The value in an array may be of **same type** or **different type** since default in shell script everything is **treated as a string**.
- Depending on the shell type, there are different ways to initialise an array such as ( ).
- To access array values, you can use **${array_name[index]}**
- To **access all the elements in an array**, you can use **@** or * character as the index of an array.

testarray.sh

```
#! /bin/sh
NAME[0]="James"
NAME[1]="Robert"
NAME[2]="John"
NAME[3]="Michael"
NAME[4]="William"

echo "The name in first index: ${NAME[0]}"
echo "The name in second index: ${NAME[1]}"

echo "All names using asterisk symbol: ${NAME[*]}"
echo "All names using at symbol: ${NAME[@]}"

OTHER=(Hello World)
echo "Other format : ${OTHER[@]}"
```

```
yspark@yspark-virtual-machine:~/Workspace/Script$ bash testarray.sh
The name in first index: James
The name in second index: Robert
All names using asterisk symbol: James Robert John Michael William
All names using at symbol: James Robert John Michael William
Other format : Hello World
yspark@yspark-virtual-machine:~/Workspace/Script$
```

Write Schell Script – Operators

- There are various operators supported by shell:
  - Arithmetic Operators
  - Relational Operators
  - Boolean Operators
  - String Operators
  - File Test Operators
- Bourne shell **didn't originally have** any mechanism to perform **simple arithmetic operations** but it uses external program, *expr*.
- The followings need to be considered while adding –
  - There must **be spaces between operators and expression**.
  - The complete expression should be enclosed between `` ,called the **backtick**.

testarithmetic.sh

```
#! /bin/sh

A=10
B=20

echo $A + $B
echo ========================
echo `expr $A+$B`
echo ========================
echo `expr $A + $B`
```

```
yspark@yspark-virtual-machine:~/Workspace/Script$ sh testarithmetic.sh
10 + 20
========================
10+20
========================
30
yspark@yspark-virtual-machine:~/Workspace/Script$
```

Write Schell Script – Arithmetic Operators

- Assume **variable a holds 10 and variable b holds 20**

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator	`expr $a + $b` will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand	`expr $a - $b` will give -10
* (Multiplication)	Multiplies values on either side of the operator	`expr $a \* $b` will give 200
/ (Division)	Divides left hand operand by right hand operand	`expr $b / $a` will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder	`expr $b % $a` will give 0
= (Assignment)	Assigns right operand in left operand	a = $b would assign value of b into a
== (Equality)	Compares two numbers, if both are same then returns true.	[$a == $b] would return false.
!= (Not Equality)	Compares two numbers, if both are different then returns true.	[$a != $b] would return true.

- It is very important to understand that all the **conditional expressions** should be **inside square braces with spaces around them**.

Write Schell Script – Relational Operators

- Assume **variable a holds 10 and variable b holds 20**

Operator	Description	Example
-eq	Checks if the value of two operands are **equal** or not; if yes, then the condition becomes true.	[$a -eq $b] is not true.
-ne	Checks if the value of two operands are equal or **not**; if values are not equal, then the condition becomes true.	[$a -ne $b] is true.
-gt	Checks if the value of left operand is **greater than** the value of right operand; if yes, then the condition becomes true.	[$a -gt $b] is not true.
-lt	Checks if the value of left operand is **less than** the value of right operand; if yes, then the condition becomes true.	[$a -lt $b] is true.
-ge	Checks if the value of left operand is **greater than or equal** to the value of right operand; if yes, then the condition becomes true.	[$a -ge $b] is not true.
-le	Checks if the value of left operand **is less than or equal** to the value of right operand; if yes, then the condition becomes true.	[$a -le $b] is true.

Write Schell Script – Boolean Operators

- Assume **variable a holds 10 and variable b holds 20**

Operator	Description	Example
!	This is logical negation. This inverts a true condition into false and vice versa.	[! false] is true.
-o	This is logical **OR**. If one of the operands is true, then the condition becomes true.	[$a -lt 20 -o $b -gt 100] is true.
-a	This is logical **AND**. If both the operands are true, then the condition becomes true otherwise false.	[$a -lt 20 -a $b -gt 100] is false.

Write Schell Script – String Operators

- Assume **variable a holds "abc" and variable b holds "efg"**

Operator	Description	Example
=	Checks if the value of two operands are **equal** or not; if yes, then the condition becomes true.	[$a = $b] is not true.
!=	Checks if the value of two operands are equal or **not**; if values are not equal then the condition becomes true.	[$a != $b] is true.
-z	Checks if the given string operand **size is zero**; if it is zero length, then it returns true.	[-z $a] is not true.
-n	Checks if the given string operand **size is non-zero**; if it is nonzero length, then it returns true.	[-n $a] is not false.
str	Checks if **str** is not the empty string; if it is empty, then it returns false.	[$a] is not false.

Write Schell Script – File Test Operators

- Assume a **variable file holds on existing file name "test"** the size of which is 100 bytes and has read, write and execute permission

Operator	Description	Example
-b file	Checks if file is a block special file; if yes, then the condition becomes true.	[-b $file] is false.
-c file	Checks if file is a character special file; if yes, then the condition becomes true.	[-c $file] is false.
-d file	Checks if file is a directory; if yes, then the condition becomes true.	[-d $file] is not true.
-f file	Checks if file is an ordinary file as opposed to a directory or special file; if yes, then the condition becomes true.	[-f $file] is true.
-g file	Checks if file has its set group ID (SGID) bit set; if yes, then the condition becomes true.	[-g $file] is false.
-k file	Checks if file has its sticky bit set; if yes, then the condition becomes true.	[-k $file] is false.
-p file	Checks if file is a named pipe; if yes, then the condition becomes true.	[-p $file] is false.
-t file	Checks if file descriptor is open and associated with a terminal; if yes, then the condition becomes true.	[-t $file] is false.
-u file	Checks if file has its Set User ID (SUID) bit set; if yes, then the condition becomes true.	[-u $file] is false.
-r file	Checks if file is readable; if yes, then the condition becomes true.	[-r $file] is true.
-w file	Checks if file is writable; if yes, then the condition becomes true.	[-w $file] is true.
-x file	Checks if file is executable; if yes, then the condition becomes true.	[-x $file] is true.
-s file	Checks if file has size greater than 0; if yes, then condition becomes true.	[-s $file] is true.
-e file	Checks if file exists; is true even if file is a directory but exists.	[-e $file] is true.

Write Schell Script – Decision Making (1)

- Shell supports **conditional statements** which are used to perform different actions based on different conditions.
- There are **two decision-making statements** available:
  - The **if…else** statement
  - The **case…easc** statement
- **if else statements** are useful decision-making statements which can be used **to select an option** from a given set of options.
- Shell supports following forms of if…else statement
  - **if…fi** statement
  - **if…else…fi** statement
  - **if…elif…else…fi** statement

testif.sh

```
#!/bin/sh

A=10
B=20

if [ $A == $B ]
then
   echo "A is equal to B"
else
   echo "A is not equal to B"
fi
```

```
yspark@yspark-virtual-machine:~/Workspace/Script$ bash testif.sh
A is not equal to B
yspark@yspark-virtual-machine:~/Workspace/Script$
```

Write Schell Script – Decision Making (2)

- The **case…esac** statement is more efficient than repeated **if…elif** statement.
- The basic syntax of the **case…esac** statement is to give an expression **to evaluate and to execute** several different statements based on the value of the expression.
- There is **no maximum number of patterns** but the **minimum is one**.
- If **no matches** are found, the case statement **exits without performing any action**.
- When statement(s) part executes, the command **;;** **indicates that the program flow should jump to the end of the entire case statement**.

testcase.sh

```
#!/bin/sh

FRUIT="kiwi"

case "$FRUIT" in
    "apple") echo "Apple pie is quite tasty."
    ;;
    "banana") echo "I like banana nut bread."
    ;;
    "kiwi") echo "New Zealand is famous for kiwi."
    ;;
esac
```

```
yspark@yspark-virtual-machine:~/Workspace/Script$ bash testcase.sh
New Zealand is famous for kiwi.
yspark@yspark-virtual-machine:~/Workspace/Script$
```

testcase1.sh

```
#!/bin/sh

option="${1}"
case ${option} in
    -f) FILE="${2}"
        echo "File name is $FILE"
        ;;
    -d) DIR="${2}"
        echo "Dir name is $DIR"
        ;;
    *)
        echo "`basename ${0}`:usage: [-f file] | [-d directory]"
        exit 1 # Command to come out of the program with status 1
        ;;
esac
```

Write Schell Script – Loops: while and until

- The following types of loops are available: **while** loop, **for** loop, **until** loop, **select** loop
- All the loops **support nesting concept** which means you can put one loop inside another similar one or different loops.
- You can use **break** and **continue** statements to control loops
- The **while loop** executes the given commands **until the given condition remain true**.
- **'while true'** or **'while :'** can make infinite loop, where colon (:) is equivalent to no operation.
- The **until loop** is similar to while loop.
- The only difference is that the **until loop** executes until a given **condition becomes true**.

testwhile.sh

```
#! /bin/bash

# Basic while loop
counter=1
while [ $counter -le 10 ]
do
    echo $counter
    ((counter++))
done
echo All done

```

testuntil.sh

```
#! /bin/bash

# Basic until loop
counter=1
until [ $counter -gt 10 ]
do
    echo $counter
    ((counter++))
done
echo All done
```

```
yspark@yspark-virtual-machine:~/Workspace/Script$ bash testwhile.sh
1
2
3
4
5
6
7
8
9
10
All done
```

Write Schell Script – Loops: until and for

- The **for loop** moves through a specified list of values until the list is exhausted.
- Syntax of for loop using **in** and **list of values**.
- There is **a three-parameter for loop control expression** like for loop in C language.
- The expression of three-parameter for loop is

  ‘for ((EXP1; EXP2; EXP3))’
- It consists of **an initialiser (EXP1)**, **a loop-test or condition (EXP2)**, and **a counting expression/step (EXP3)**.
- The example shows the same result using in with a list and **three-parameter for loop**.

testfor.sh

```
#!/bin/bash

# Basic for loop
names='Stan Kyle Cartman'
for name in $names
do
    echo $name
done
echo All done
```

```
yspark@yspark-virtual-machine:~/Workspace/Script$ bash testfor.sh
Stan
Kyle
Cartman
All done
yspark@yspark-virtual-machine:~/Workspace/Script$
```

testfor1.sh

```
#!/bin/bash

for (( i = 1; i <= 5; i++ ))
do
    echo "1. Welcome $i times"
done

echo ==============================

for i in 1 2 3 4 5
do
    echo "2. Welcom $i times"
done
```

Write Schell Script – Loops: select

- **The select** loop provides an easy way to **create a numbered menu** from which users can select options.
- It is useful when it needs **to ask the user to choose one or more items** from a list of choices.
- The list of choices is sequences of characters **separated by space**.
- It is possible to **display a prompt** using the variable **PS3**.
- You will have to make sure **when the select loop exists** by checking a condition using **if** or **case** statements.
- If the condition is satisfied, use **break** statement to exit the loop.

testselect.sh

```
#! /bin/bash
# A simple menu system
names='Kyle Cartman Stan Quit'
PS3='Select character: '
select name in $names
do
    if [ $name == 'Quit' ]
    then
        break
    fi
    echo Hello $name
done
echo Bye
```

```
yspark@yspark-virtual-machine:~/Workspace/Script$ bash testselect.sh
1) Kyle
2) Cartman
3) Stan
4) Quit
Select character: 1
Hello Kyle
Select character: 2
Hello Cartman
Select character: 3
Hello Stan
Select character: 4
Bye
```

Write Schell Script – Functions

- Functions enable to **break down** the overall functionality of a **script into smaller**, logical subsection.
- **Using functions** to perform repetitive tasks is an excellent way to **create code reuse**.
- Functions have to **be declared** in the shell script **before using it**.
- You can define a function that will **accept parameters while calling the function**.
- You can **return any value** from a function using **return** command.
- It is possible to call themselves and another function, known as a **recursive function**.

testfunction.sh

```
#!/bin/sh

# Define your function here
Hello () {
    echo "Hello World $1 $2"
    return 10
}

# Invoke your function
Hello Zara Ali

# Capture value returnd by last command
ret=$?

echo "Return value is $ret"

```

```
yspark@yspark-virtual-machine:~/Workspace/Script$ bash testfunction.sh
Hello World Zara Ali
Return value is 10
yspark@yspark-virtual-machine:~/Workspace/Script$
```

Write Schell Script – Functions: Parameters

- To pass any number of parameters to a function, you are required to **insert them just after the function's name**.
- You must **apply spaces** between function name and arguments.
- It might be a good option to use **double quotes** in a case any parameter has spaces.
- The given parameters are accessed as $1, $2, $3 ... **corresponding to the position** of the parameters.
- The **$0** variable is kept reserved for the function name.
- The **$#** variable is used to hold **the number of parameters** given to the function.
- The **$*** and **$@** are used to hold **all the parameters** given to the function.

testfunction.sh

```
#!/bin/sh

# Define your function here
Hello () {
   echo "Hello World $1 $2"
   return 10
}

# Invoke your function
Hello Zara Ali

# Capture value returnd by last command
ret=$?

echo "Return value is $ret"

```

```
yspark@yspark-virtual-machine:~/Workspace/Script$ bash testfunction.sh
Hello World Zara Ali
Return value is 10
yspark@yspark-virtual-machine:~/Workspace/Script$
```

Summary

- **Linux** is a family of **open-source Unix-like operating systems** based on Linux kernel.
- **Linux distribution** is a **collection of applications, packages, managements, and features** that run on top of the **Linux kernel**.
- There are many Linux distribution such as **Arch**, **CentOS**, **Debian**, **Fedora**, **OpenSUSE**, **Ubuntu**, etc.
- Linux runs **faster** even with older hardware but Windows are **slower** compared to Linux.
- Linux provides a **command line interface** for interactive access to operating system **functions and services**.
- The command line interface is **very useful in a case no graphical user interface**. So, it is better to remember basic commands such as ls, cd, cp, mv, rm, mkdir, rmdir, touch, cat, pwd, history, grep, find, head, tail, etc.

Source: Shutterstock

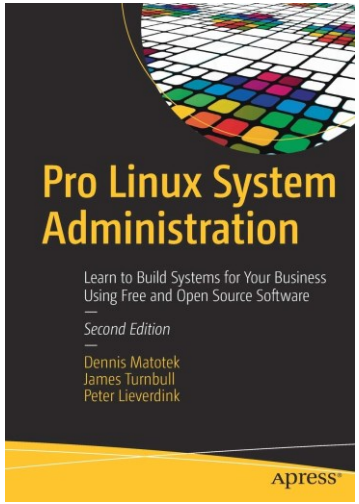
Summary

- **Linux** was designed to **allow more than one user** to have access to the system at the same time.
- Each file and directory on Linux system **is assigned 3 types of ownership**: **User**, **Group**, and **Other**.
- The **systemd** process is **the base process** on Linux and **must be running** for a host to be functional.
- Package management is **a method of installing and maintaining software** on a host. The most common way to add software on Ubuntu hosts is to install packages via **command-line tools such as apt, dkpg, aptitude**.
- **Shell script** is a computer program **designed to be run by the Unix/Linux shell**.
- Shell provides users with an interface like **accepts human-readable commands** into the system; **executes those commands**; **give the program's output**.

Source: Shutterstock

REFERENCE

[Main References]



Pro Linux System Administration
Dennis Matotek
James Turnbull
Peter Lieverdink

Part I: The Beginning

[Other References]

- Wikipedia:
  - https://en.wikipedia.org/wiki/Linux
  - https://en.wikipedia.org/wiki/Shell_script
  - https://en.wikipedia.org/wiki/Linux_distribution
- Guru99:
  - https://www.guru99.com/install-linux.html
  - https://www.guru99.com/introduction-linux.html
  - https://www.guru99.com/linux-differences.html
- TutorialsPoint: https://www.tutorialspoint.com/unix/shell_scripting.htm
- Ryans Tutorials: https://ryanstutorials.net/bash-scripting-tutorial/
- Linux.com: https://www.linux.com/training-tutorials/understanding-linux-file-permissions/
- HOSTINGER: https://www.hostinger.co.uk/tutorials/linux-commands

THANKS!

Any questions?

y.s.park@warwick.ac.uk