Strengths and Weaknesses of Ubuntu

The key strengths of Ubuntu are usability, good device support, support of large user and developer communities and adoption by many manufacturers as well as some large organizations. A constant complaint against Ubuntu is that it does not give back to the community as much as it takes from it and often deviates from common interests to chart its own path. Canonical's commercial interests are also sometimes a topic of debate.

The Present Landscape of Unix / Linux Derivatives

Many derivatives of Unix and a large number of distributions of Linux are in use today. The common Unix derivatives include BSD Unix, FreeBSD, OpenSolaris, HP-UX, etc. There are many Linux distros and their popularity is tracked by websites such as distrowatch.com.

Pros and Cons of using Linux

Linux is a free and open source operating system that gives its users all the freedoms. It is known for its performance, security, scalability and portability. it provides an excellent environment for development. It provides a powerful command line environment as well as an easy to use graphical environment. It also has good support for the PC hardware. There is a large community of users who are ready to offer help without expecting anything in return. it has a vast collection of free and open source software that makes it a perfectly usable system for most use cases. Its high performance, scalability and security make it an ideal choice for servers. Due to its ability to work remotely through command line, availability of powerful open source development platforms and low hardware requirements have made Linux the operating system of choice for cloud deployment also.

On the other hand, naive end users often complain that Linux is not as intuitive or easy to use as its proprietary alternatives. While the community provides good support for major issues, minor issues are often relegated to the back burner. Paradoxically, freedom provided by open source software is also a problem. Software changes too fast, leading to a rather steep learning curve. There is too much choice for some types of software and the preferences continually shift.

Linux is a good platform for those people who have an ideological preference for freedom and openness and who are willing to put in a little effort in learning new things. It is also a very good platform for servers, development systems and cloud deployments.

Pros and cons of using Linux

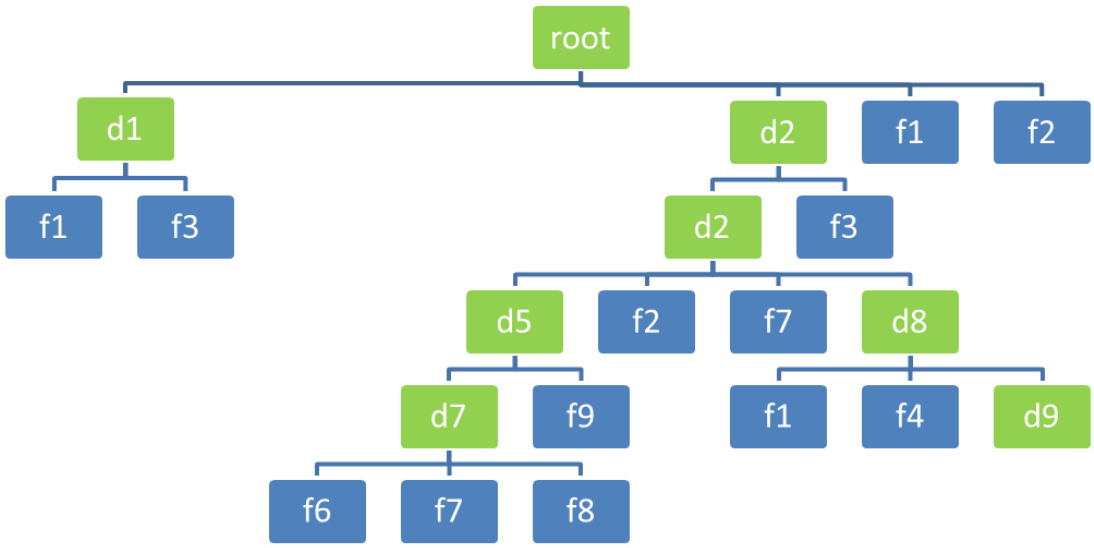
- Cost
- Flexibility
- User-friendliness
- Power
- Compatibility with hardware, device drivers
- Compatibility with Windows software (wine, mono)
- Security
- Collaborating with Windows-based PCs and Servers
- Developed for the programmers by the programmers (CLI)
    - Assumes that you know what you are doing

The File System

The general purpose computers also have storage facilities. The secondary storage is organized in blocks and, like everything else with computers; the blocks are accessed by block numbers. Today's hard disks may have over a billion blocks. This method, though natural for computers, is quite problematic for humans. It is impossible for a human being to remember what was stored in which block number. When some piece of information is large requiring many blocks, one must remember a series of block numbers (that need not be sequential). There is no marking on the blocks to indicate which one is in use and which one is not. One must be careful not to commit the mistake of writing some data to a block to which earlier some other useful data was written; as in that case the previously written data would be silently overwritten and lost. In case of the same system being used by multiple users, of course, there is no way for one user to know which blocks are in use by the other users.

To avoid all these problems, the operating system provides a file system interface to secondary storage. The concept of a file system is modeled after the filing cabinets commonly found in offices, but with new twists. A file system chiefly contains two types of objects – files and directories (also known as folders). A file is the basic unit of secondary data storage on computers. Any data that the user wants to store will go in some file in the file systems. The files are identified by their names, which are much easier for humans to remember than block numbers. As a disk may have a large number of files, directories are used to organize them. A directory is nothing but a container that may contain files as well other directories known as its subdirectories. The subdirectories may, in turn, contain files as well as subsubdirectories and these may contain files as well as further directories and so on. In fact, there is no theoretical limit to such nesting (putting one container object inside another). However, every file system starts with what is called its root directory and then the

root directory may contain files as well as subdirectories and it can go on and on like that. The following figure illustrates this concept. Here the green nodes represent directories, while the blue ones represent files. As an added convenience, directory names begin with d (except root) and filenames begin with f, though there is no such rule.



The file system structure

Internally, the data is still accessed by disk block numbers, but now the file system component of the operating system keeps track of which file or directory is stored in which block number(s) and which blocks are free and which ones are in use. This takes a great burden off the human users. To maintain this information, some data structures are created on the disk.

The file systems follow certain basic rules. Each file system has a single root directory that is the starting point of the file system. Each directory in the file system contains a number of objects (files and directories), each of which must have a name unique within that directory. Thus, there can never be two objects with the same name in the same directory. However, two different directories can have two different objects with the same name. As long as you know the directory containing the object you want to use, knowing its name may be enough as it is guaranteed to be unique within its directory. Otherwise, you will have to specify the absolute (or full) path leading to the object. That is, you start with the root directory, then the subdirectory, then the sub-

subdirectory, and so on, until you reach to the directory that contains the object in question and finally the object itself. Each of these components is separated by a special character that is not permitted in file or directory names. This path uniquely identifies an object in the entire file system.

The file system structure is generally described as a tree structure, with the root directory forming the only root of the tree, all other directories forming branches and the files forming the leaves. If a directory is contained inside another directory, the former is said to be the child of the latter and the latter is said to be the parent of the former. Every directory, except the root directory, has a parent directory. The files cannot hold further objects (files or directories) in them, and hence are the end of the branches. They are known as leaves. With computers, tree structures are typically drawn upside down, with the root at the top.

A blank disk initially contains no file system. The initial blank file system structure (empty data structures) is created by an operation known as formatting the disk. Formatting a disk that already contains a file system destroys the existing file system and replaces it with a new blank one. Certain types of disks such as the hard disks can contain multiple partitions. Each partition may be formatted to contain a separate and independent file system. Formatting one partition does not affect the others. The operating system provides utilities for viewing, modifying and formatting the disks and their partitions.

The command line interface provides commands to navigate the directory structure, while the graphical environment provides a file browser to explore the file system. Common operations provided at the file system level with both the user interfaces include creating new files and directories, copying a file or directory to another part of the file system, moving a file or directory to another part of the file system, renaming a file or directory, deleting (removing) a file or directory (including all the contents), etc. Often security constraints may be in place to prevent a user from carrying out certain operations on certain files or directories.

Different devices and different operating systems use different file systems. For example, Microsoft Microsoft Windows family of operating systems typically use a file system called FAT (File Allocation Table) along with its variants and a file system called NTFS (New Technology File System). The first one is very simple in nature, but limited in power and performance. It also does not provide any security. The latter is far more advanced than the former and provides a host of features, including security. Both file systems are case insensitive, i.e. capital letters and small letters are treated as same. Thus, you cannot have two files with the names myletter and MyLetter in the same directory. Both generally divide the filename into two parts – the filename proper followed by a . (dot) and the extension. Both use the \

(backward slash or backslash) character as the path separator character. The extension part of the filename is used to signify the type of the file. Different Unix systems use different file systems. Linux generally uses some version of the extended file system (ext2, ext3 or ext4), though several others are also available and in use. These are quite powerful and feature rich file systems. The extended file system is case sensitive, i.e. capital and small letters are treated as two different characters. So you may have two files with the names f1 and F1 in the same directory. It uses the / (slash) character as the path separator. The concept of using the extension part of the filename to signify its type is not mandatory and is weakly used. USB flash disks and memory cards used with mobile phones usually come formatted with the FAT file system when the size is upto 32GB and the exFAT file system when the size is > 32GB (for example, the micro SDXC cards). CDs generally use the ISO9660 file system, while DVDs use the UDF file system. These different file systems have different characteristics.

The Unix/Linux File System

- Hierarchical file system
    - While the Windows file systems is actually a forest, the UNIX file systems is a proper tree (actually a directed acyclic graph)
- The root directory is identified by the / (slash) character
- Directories and files
- Identifying a file system object uniquely using path
- Components of the path are separated by the / (forward slash) character
- / (forward slash) v/s \ (backslash)
- Filenames are case sensitive
- Extensions not an essential part of file name (executable files usually have no extension)
- Any character except / can be used in file names
- The notion of current directory and relative paths
- Absolute path v/s relative path
- Interactive users have a home directory. A user has full permissions on one's home directory. The ~ (tilde sign) is a shortcut for the user's home directory in file system manipulation commands
- Hidden files and directories - any file or directory whose name starts with a . is considered hidden. Hidden files and directories are not displayed in the nautilus file browser (GUI) or by the ls command (CUI). However, there are options to display them

Accessing Multiple File Systems

A computer system may have many storage devices in it. Also, removable devices may be inserted or attached and removed at any time. Each device has its own file system on it. A device that can have multiple partitions, like the hard disk, has a separate file system on each partition. How does one access these file systems? Operating systems like Microsoft Windows assign a separate drive letter (like C:, D:, E:, etc.) to each file system. However, Linux and other Unix-like systems have a single file system tree starting with the root directory, denoted by / (the slash character). The file system contained on the partition from which Ubuntu boots is called the root file system. The root directory of this file system becomes / - the root of the entire file system tree. Initially this is the only file system available.

We may access any other file system by mounting it on any existing directory (this directory is called the mount point). Once mounted, the contents of that file system appear as the contents of the mount point directory. If the mount point previously contained some contents (files and subdirectories), they are masked (hidden) for the duration of the mount. Now we may access (and modify) the contents of that file system from the mount point directory. When we no longer need to use the file system, we may unmount it. At this point, the original contents of the mount point directory get unmasked (become visible again). This process is depicted in the following figures a, b, c and d. Figure a shows the root file system. Figure b shows the file system on another device. Figure c shows the situation after mounting the file system of figure b onto the directory d3 of Figure a. The original contents of d3 are now masked and the contents of the file system mounted there appear as if they are the contents of d3. Figure d shows the situation after unmounting the second file system. The original contents of the directory d3 now become visible again.

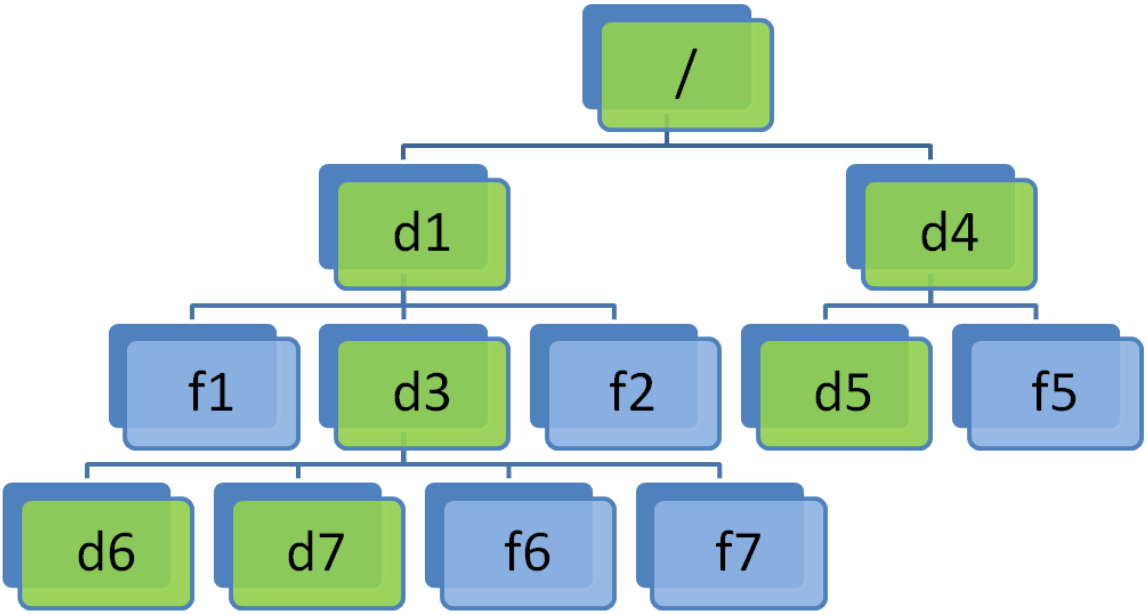


Figure a: The root file system

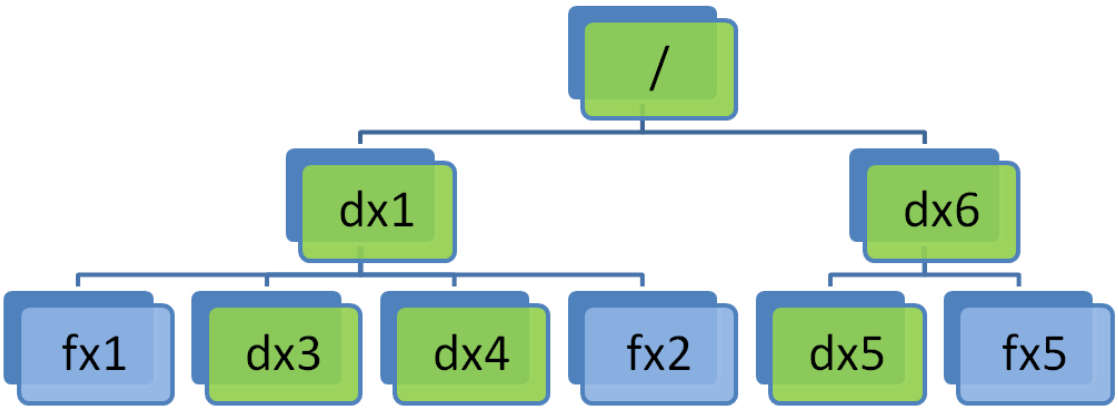


Figure b: File system on another device

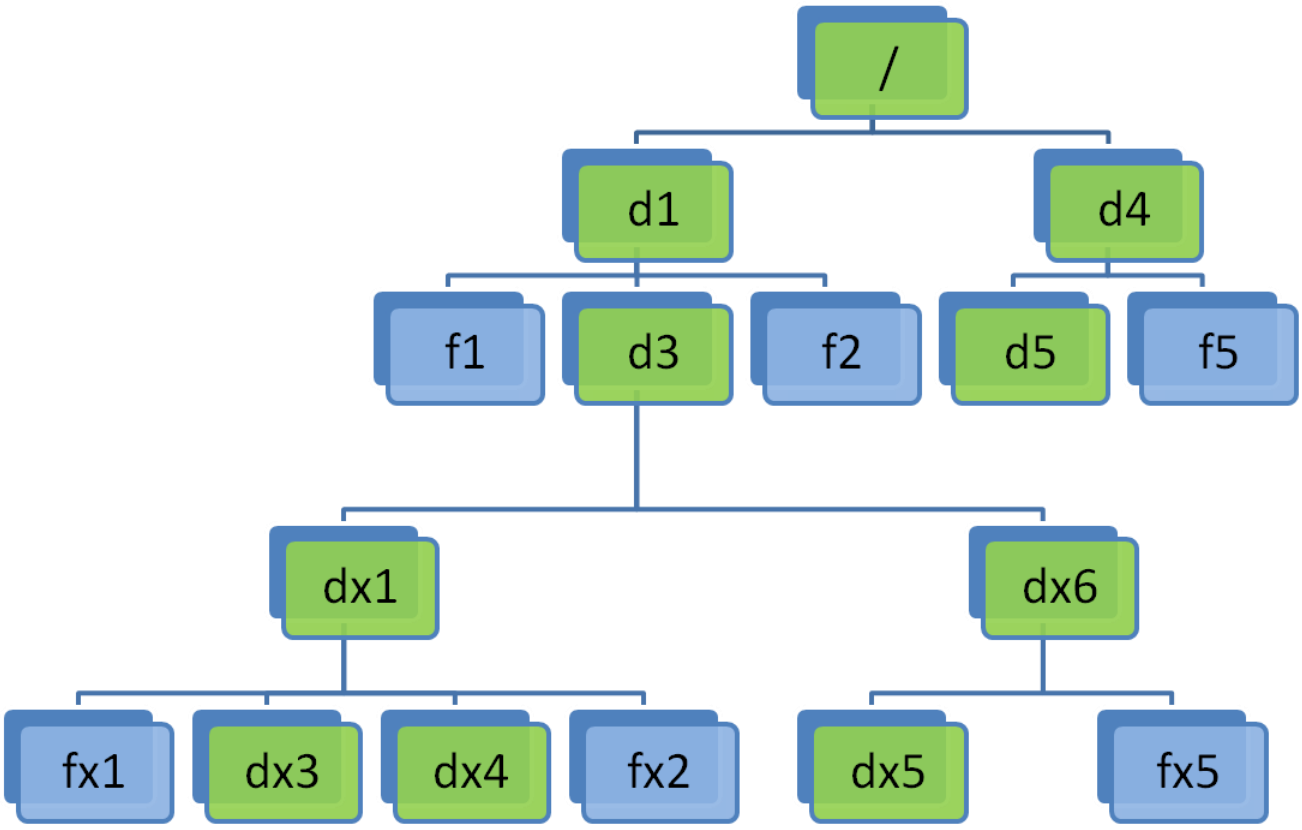


Figure c: After mounting the second file system on directory d3

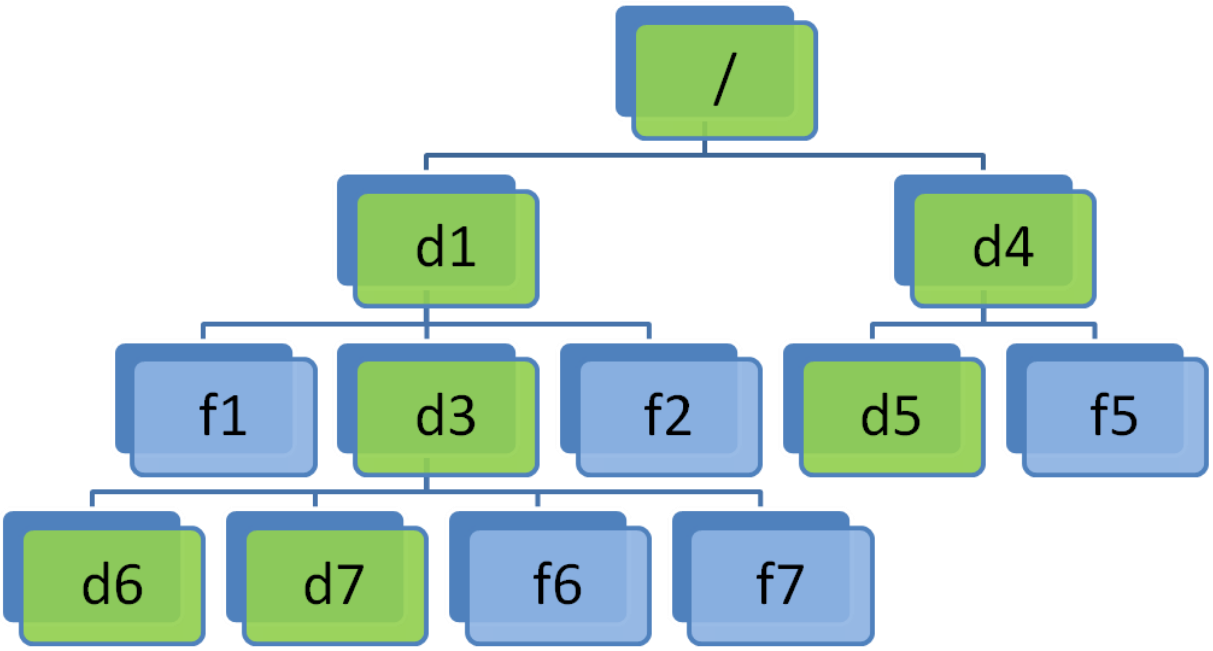


Figure d: After unmounting the second file system from directory d3

However, the common practice is to mount file systems onto empty directories. In the default configuration, Ubuntu automatically detects other fixed devices in the system and shows them in the places menu and the left pane of the file browser. They are mounted when we first try to access them. The file browser shows a triangular icon alongside all mounted file systems other than the root file system. The file system can be unmounted by clicking on this icon. It may be mounted again when the user tries to access it again next time. The root file system cannot be unmounted. Removable devices are automatically mounted when they are inserted. Read-only media like optical disks can be unmounted by simply removing or ejecting them. Media on which writing is possible (like USB flash disks) must be unmounted by clicking the unmount icon in the file browser or by right clicking its icon on the desktop and selecting "safely remove device" option. This causes any data cached in main memory for improved performance to be flushed to the disk. A message at the end of this process announces that it is now safe to remove the device and the unmount icon disappears from besides the device's entry in the file browser's left pane. Only after this the device can be safely detached from the system. Failure to observe this procedure may result in loss of data or damage to the file system on the removable media. While this method of accessing the other storage devices in the system may sound unnecessarily complicated, it is more flexible and powerful and has several distinct advantages.

Not only local file systems, even the remote file systems can also be accessed in the same way. For example, Windows shares can be mounted on local directories using the SMB (Server Message Block) protocol and then be accessed just like local content. Similarly, NFS (Network File System), FTP server directories can also be mounted on a directory and accessed as a part of the file system.

Of course, all these procedures can also be carried out using commands. It is also possible to configure the system to mount some file systems at particular mount points automatically every time the system boots. By default, the system mounts other fixed and removable devices in the system in directories under the /media directory.

User Accounts and Home Directories

There are different kinds of users in the Linux system. *Normal* users can control their own files and settings, but cannot make system level changes. *Administrative* users can access files belonging to other users also. They can also make system level changes. An *interactive* user is a user who can login to the system through the (text mode or graphical) terminal interactively. A *non-interactive* user account is created and used by the operating system or some other software package internally. Such users are not allowed to login to the system interactively. Every interactive user has a home directory. A user has full permissions on one's home directory while other normal users have no permissions on it (a user with administrative permission can still access it). In paths, the ~ (tilde) character represents the currently logged-in user's home directory

User accounts are identified by their names by the human users of the system; but internally, they are identified by an integer user id. There is a special user account with numerical user id 0, usually called *root*. This user is the most powerful user in the system.

The Standard File System Layout

- **/bin** (binary) This directory contains the standard programs that may be useful to all the users of the system
- **/boot** This directory contains the files used for booting the system
- **/dev** (device) This directory contains the files corresponding to various hardware devices
- **/etc** (et cetera) This directory contains all the system-wide configuration files
- **/home** This directory contains the home directories of interactive users (except *root*)
- **/lib** This directory contains the system libraries
- **/lib32** This directory contains the 32-bit system libraries on a 64-bit system

- **/lib64** This directory contains the 64-bit system libraries on a 64-bit system
- **/media** This directory is used to mount removable drives automatically when they are inserted
- **/mnt** This directory is used for additional file systems that are to be mounted on a permanent basis (at boot time)
- **/opt** On some Unix-like systems, this directory is used to store optional / additional software installed by the user
- **/proc** This directory contains the virtual (not real) proc file system, which is used to access live kernel parameters
- **/root** This directory is the home directory of the root user
- **/sbin** this directory contains the system programs useful to system administrators
- **/tmp** This directory contains a virtual file system that exists in RAM. It loses its contents upon reboot. Many software programs use this directory for creating temporary files
- **/usr** This directory holds the additional software installed by the user. It has a substructure that mirrors the structure of / (the root directory) to some extent
    - **/usr/bin** This subdirectory contains the executable programs for user-installed software
    - **/usr/include** This directory contains the header files included in C/C++ programs
    - **/usr/lib** This directory contains the libraries for the user-installed programs
    - **/usr/local** This directory contains programs for local use only (as opposed to use from a remote terminal). It also has a substructure similar to /usr
    - **/usr/man** This directory contains the pages of the online manual
    - **/usr/share** This directory contains shared files belonging to different programs
- **/var** This directory contains frequently changing data, including cache files, printer spool files as well as log files
    - **/var/log** This directory contains the various system and software logs. these logs are useful for debugging problems and to audit the system security
    - **/var/www/html** or **/var/www** this directory is the Apache web server web-root and only exists id Apache is installed

The Shells

Unix offers a variety of shells in both text mode and graphical mode. The graphical mode shells include the GNOME shell, Unity, KDE, XFCE, LXDE, MATE, etc. The

commonly used text mode shells include sh (the original Bourne shell), ksh (Korn shell) csh (C Shell), bash (Bourne-Again SHell), etc.

The Graphical Shell (GNOME Shell / Unity)

Switching between programs

- ALT-TAB to switch between programs
- ALT-` (back-quote) to switch between windows of the same program

Multiple screens

- 12 virtual screens (6 text mode screens and 6 graphics mode screens) accessed using the shortcut keys CTRL-ALT-F1 ... CTRL-ALT-F12
- Each graphical screen can be configured to have multiple workspaces. Workspaces are arranged in a rectangular grid with some number of rows and some number of columns. To switch between workspaces and to drag and drop windows between workspaces, one may use the *Workspace Switcher*. Shortcut keys for switching between the workspaces are CTRL-ALT-Arrow Keys. To move a window to a different workspace, press ALT-SPACE to open the window menu and select appropriate option; or open *Workspace Switcher* and use drag and drop
- In the graphical mode, we may open one or more windows of the GNOME Terminal program to work in text mode. In each terminal window, we may have one or more tabs. The shortcut key for creating a new tab is CTRL-SHIFT-T and the shortcut keys for switching between tabs are CTL-PgUp and CTL-PgDown

The Bourne Shell

- **Internal commands** are built into the shell itself. There is no seperate executable file for them
- **External commands** are not built into the shell. There is a seperate executable file for them
- Search path
    - Since it is not practical to search the entire hard disk for the executable file corresponding to a possible external command entered by the user, only the directories in the search path are searched for the same. The search path is stored in the built-in shell variable PATH as string containing one or more directories seperated by the : (colon) character
    - The current directory is not searched for a command if it is not in the search path

- CTRL-D at the beginning of a line indicates end-of-file (or end-of-input)
- CTRL-C is used to terminate the currently rnning foreground process
- CTRL-S is used to pause the terminal
- CTRL-Q is used to resume the terminal
- To copy text from the terminal, select the text using the mouse and press CTRL-SHIFT-C
- To paste text into the termnal, press CTRL-SHIFT-V. The text is inserted as if it had been typed by the user
- The shell maintains the history of the commands entered earlier in memory. When the shell session terminates, this command history is saved in the file ~/.bash_history. The command history can be accessed using the UP / DOWN arrow keys at the prompt. It is also available in subsequent sessions and survives reboots
- The shell has its own full-fledged built-in programming language with variables, control structures, input, output, etc. It is a dynamic language in nature
- Case sensitivity
- Command completion using TAB and TAB-TAB

The Command Line

- components of the command line and word splitting: The command line contains one (or more) command(s), arguments to the command, operators, etc. The shell performs word splitting on the command line using white space (space / tab characters) as the seperators
- In general, order of compnents of a command line is not important. However, there are several exceptions
- Options
    - **Short options** Short options consist of - (hyphen) followed by a single character
    - **Long options** Long options are specified using -- (two consecutive hyphens)
    - **Options with arguments** Some options have their own arguments. in such cases, the argument(s) must immediately follow the option
    - **Combining short options** Multiple short options can be combined by writing them immediately after a single hyphen: `ls -lh`. Only the last short option can have argument(s) in such case; and the argument, if any, is immediately specified after the combined options: `tar -czf backup.tar.gz directory`
- **Quoting and escaping** Strings on the command line can be specified without using any quotes. However, if a string contains white space or some other

character having a special meaning to the shell interpreter, the string must be enclosed in quotes or the white spaces / special characters must be escaped
    - **Escaping** A character having special meaning to the shell can be escaped by preceeding it with \ (backspace). This removes its special meaning. The special meaning of the backslash itself can also be removed by preceeding it with another backslash `\\`
    - **Single Quotes** Almost no pocessing is carried out inside strings enclosed in singl quotes `'aaa bbb ccc'`
    - **Double Quotes** Double quotes also behave like single quotes, but variable / parameter expansion *is* carried out inside double quotes: `$variable_name` is converted into the value of the variable. Hence if the value of variable *name* is *Scotty*, `echo 'Beam me up, $name'` outputs `Beam me up, $name` but `echo "Beam me up, $name"` outputs `Beam me up, Scotty`
- Operators (Input/Output Redirection, Combining Multiple Commands using Logical Operators)

File System Manipulation Commands

Note: *File system object* means either a file or a directory

- **pwd** (present working directory)
    - **pwd** Outputs the current working directory
- **cd** (change directory)
    - **cd directory** Changes the current directory to the given directory
    - **cd** changes the current directory to the home directory of the user. This behaviour is different from Windows where `cd` displays the current directory
- **ls** (list)
    - **ls** Outputs the names of the file system objects in the current directory
    - **ls arguments** Outputs the information about the argument(s). In case an argument is a file, its name is displayed. In case an argument is a directory, the names of file system objects inside the directory are displayed. If the argument file system object does not exist, an error message is output
    - **Options**
        - **-l** (*long* listing) Outputs a long listing with more information about the file system objects
        - **-d** (*directory*) When an argument is a directory, outputs information about the directory itself rather than its contents

        - **-R** (*recursive*) Outputs information about each argument along with child file system objects recursively
- **mkdir** (make directory)
    - **mkdir directory** Creates the given directory if it does not exist
- **rmdir** (remove directory)
    - Removes (deletes) the given directory if it is empty
- **cat** (concatenate)
    - **cat file(s)** Outputs a concatenation of the given files. Also used to output a single file
- **cp** (copy)
    - **cp arguments** There must be at least two arguments. The last argument is the target. All penultimate arguments are sources. If the target is a directory, the sources are copied into the target. If the target is a file, there must be only one source and the source file is copied onto the target file. If the target does not exist, it is assumed to be a file and is created. *cp* does not copy directories by default. There are no warning or prompts if a file is going to be overwritten
    - **Options**
        - **-r** (recursive) This option is used to copy an entire file system tree rooted at the given node (i.e. a directory along with all its contents recursively)
        - **-i** (interactive) Prompts before overwriting a file
- **rm** (remove)
    - **rm arguments** Removes (deletes) the given arguments. While the files and directories deleted from the GUI go into the trash, files and directories deleted from the command prompt are permanently lost. Does not delete directories by default. Does not prompt before deleting a file
    - **Options**
        - **-r** (recusrive) Deletes the arguments reursively. Directories in the arguments are deleted with all the contents recursively
        - **-i** (interactive) Prompts before deleting a file
- **mv** (move)
    - **mv arguments** Moves or renames as per the number and types of arguments
    - **How is the target of the move is determined**
        - If the destination is the name of an existing file, the target of move is assumed to be that file
        - If the destination is the name of an existing directory, the target is assumed to an object with the same name as the source inside that directory

        - If the destination does not exist, the destination name itself is the target. It is considered a file name if the source is a file and a directory name if the source is a directory
    - **When a move operation is performed**
        - If the directory containing the source and the directory containing the target are different, the source object is moved
    - **When a rename operation is performed**
        - If the source name and target name are not same, a rename operation is performed
    - **When a file is overwritten**
        - If the target is a file and it already exists, it is overwritten
    - **Operations that are not allowed**
        - If there are more than one sources and the destination is not a directory
        - If the source is a directory and the target is an existing file
        - If the source is a filename and the target refers to the same file
    - **Options**
        - **-i** (interactive) Prompts before overwriting a file
- **dirs** (directories)
    - **dirs** Displays the current directory stack
- **pushd** (push directory)
    - **pushd directory** Pushes the directory onto the directory stack and then changes to that directory
- **popd** (pop directory)
    - **popd** Pops the top-of-the-stack directory from the dirctory stack and changes to the new top-of-the-stack directory

Shell Globbing Patterns

Pattern	Matches with
?	Any single character
*	Any number of any characters
[abcde]	Any single character from among a, b, c, d and e
[^abcde]	Any one character other than a, b, c, d and e
[aeiou]	Any single vowel
[^aeiou]	Any one character other than a vowel
[abc^de]	Any one character from among a, b, c, ^, d and e
[a-z]	Any single lower case alphabetic character
[a-zA-Z]	Any single lower or upper case alphabetic character

Pattern	Matches with
[a-zA-Z_%]	Any single character from among alphabetic characters, _ and %
[a-zA-Z0-9]	Any single alphanumeric character
[^a-zA-Z0-9]	Any single non-alphanumeric character

Examples

Pattern	Matches with
p*	Anything starting with p
p*e	Anything that starts with p and ends with e, including pe
*~	All backup files
b??h	b, followed by exactly two characters, followed by h
b*h	Anything that starts with b and ends with h, including bh
dir*	Anything that starts with dir, including dir
dir?	dir followed by exactly one character

Plain Text Editors

- Plain text editors
  - Graphical editors
    - gedit / Text Editor (default)
    - gvim
    - leafpad
  - Text mode editors
    - vi (default)
    - vim (VI iMproved)
    - nano, pico
    - emacs (Editing MACroS)

Brief Overview of Working with the Vim Editor

- vim stands for VI iMproved
- It is an improved version of the vi editor
- vim has several *modes*
  - *Normal* (*command*) mode
    - When vim starts, it is in this mode
    - In this mode, the keys pressed by the user are interpreted as commands and corresponding actions are taken
  - *Insertion* mode

        - This mode is entered by pressing the *i* command in the command mode. Several other commands also switch to insertion mode
        - in this mode, the keys pressed by the user are treated as characters to be inserted into the text being edited at the curent cursor position
        - One may switch back to the command mode by pressing `ESC` in the insertion mode
    - *Last line* mode
        - This mode is entered when using certain commands in the command mode, like `:`
        - It is used for *ex commands* and long commands
        - The cursor moves to the last line where the rest of the command is entered
        - The command is completed (and executed) by pressing `ENTER` or `ESC`
        - To cancel the command, press `CTRL+C`
- Cursor movement keys like the arrow keys, HOME, END, PgUp, PgDn, etc. work as expected in both the command mode and the insertion mode in vim
- When vim is installed, the command *vi* also invokes *vim* only
- A file is opened by passing its name on the command line

```
vi filename
```

- A file is saved by typing `:w` (write) in the command mode
- To save the current file and close vi, type `:wq` (write and quit) in the command mode
- To close vi without saving the current file, type `:q!` (quit, with force) in the command mode
- An entire line can be deleted by pressing `dd` in the command mode
- *n* lines can be deleted by pressing *n*`dd` in the command mode. Here *n* is an integer and is known as the repeat count
- Deleting line(s) actually *cuts* them, so they can be pasted immediately afterwards
- `yy` (yank = copy) copies a line of text
- *n* lines can be copied by pressing *n*`yy` in the command mode.
- Text that is copied or cut (deleted) can be *put* (pasted) after the cursor using the command `p` (put) and before the cursor using the command `P`
- `u` (undo) can be used to undo the operations, while `CTRL-R` (redo) can be used to redo the operations that were previously undone