**Assignment 1**

* **Introduction to O.S.**

[An](https://simple.wikipedia.org/wiki/An) **operating system** (or **OS**) is the name for a group of [computer programs](https://simple.wikipedia.org/wiki/Computer_program),  [device drivers](https://simple.wikipedia.org/wiki/Device_driver), [kernel](https://simple.wikipedia.org/wiki/Kernel_(computer_science)), and other things that let a [user](https://simple.wikipedia.org/w/index.php?title=User&action=edit&redlink=1) work with a [computer](https://simple.wikipedia.org/wiki/Computer). It can be small (like [MenuetOS](https://simple.wikipedia.org/wiki/MenuetOS" \o "MenuetOS)), or big (like [Microsoft Windows](https://simple.wikipedia.org/wiki/Microsoft_Windows)). Different operating systems can be used for different purposes. Some are used for every day things like on a [personal computer](https://simple.wikipedia.org/wiki/PC). Others are used for [specialized](https://simple.wikipedia.org/w/index.php?title=Specialize&action=edit&redlink=1) work.

An operating system has many jobs. It is responsible for making sure that all the programs can use the [CPU](https://simple.wikipedia.org/wiki/CPU), system [memory](https://simple.wikipedia.org/wiki/RAM), displays, input, and other [hardware](https://simple.wikipedia.org/wiki/Computer_hardware). It also lets the user have a fast, clean, and safe [interface](https://simple.wikipedia.org/wiki/Interface) so they can do work on the computer. It also talks to other computers or devices on a [network](https://simple.wikipedia.org/wiki/Network).

**History**

The first operating system was used with the [ENIAC](https://simple.wikipedia.org/wiki/ENIAC) (Electronic Numerical Integrator and Computer).[[1]](https://simple.wikipedia.org/wiki/Operating_system#cite_note-pkrz_rutgers-1) It was very hard to make ENIAC do work. How the operating system worked was based on how the [switches](https://simple.wikipedia.org/wiki/Switch) and [cables](https://simple.wikipedia.org/wiki/Cable) were put together and depending on this factor [punch cards](https://simple.wikipedia.org/wiki/Punch_card) would make a [result](https://simple.wikipedia.org/w/index.php?title=Result&action=edit&redlink=1). While this was an operating system of a kind, it is not what is thought of as one in modern times.

The first operating system that looked and felt like operating systems in the modern age was [UNIX](https://simple.wikipedia.org/wiki/UNIX), made in 1969 by [Bell Labs](https://simple.wikipedia.org/wiki/Bell_Labs). It had a small kernel and many tiny programs that could be put together to work with user [input](https://simple.wikipedia.org/wiki/Information) and [data](https://simple.wikipedia.org/wiki/Data). Many of its [features](https://simple.wikipedia.org/w/index.php?title=Feature&action=edit&redlink=1) were taken from [Multics](https://simple.wikipedia.org/w/index.php?title=Multics&action=edit&redlink=1), an older operating system made in 1964.

* Linux

Computer [operating system](https://en.wikipedia.org/wiki/Operating_system) assembled under the model of [free and open-source software](https://en.wikipedia.org/wiki/Free_and_open-source_software) development and distribution. The defining component of Linux is the[Linux kernel](https://en.wikipedia.org/wiki/Linux_kernel), an [operating system kernel](https://en.wikipedia.org/wiki/Kernel_(computing)) first released on September 17, 1991 by [Linus Torvalds](https://en.wikipedia.org/wiki/Linus_Torvalds). The [Free Software Foundation](https://en.wikipedia.org/wiki/Free_Software_Foundation) uses the name [GNU](https://en.wikipedia.org/wiki/GNU)/Linux to describe the operating system, which has led to some [controversy](https://en.wikipedia.org/wiki/GNU/Linux_naming_controversy)

Linux was originally developed for [personal computers](https://en.wikipedia.org/wiki/Personal_computer) based on the [Intel x86](https://en.wikipedia.org/wiki/Intel_x86) architecture, but has since been [ported](https://en.wikipedia.org/wiki/Porting) to more [platforms](https://en.wikipedia.org/wiki/Computer_hardware_platforms) than any other operating system. Because of the dominance of the Linux kernel-based [Android](https://en.wikipedia.org/wiki/Android_(operating_system)) OS on [smart phones](https://en.wikipedia.org/wiki/Smartphone), Linux has the [largest](https://en.wikipedia.org/wiki/Usage_share_of_operating_systems) [installed base](https://en.wikipedia.org/wiki/Installed_base) of all [general-purpose operating systems](https://en.wikipedia.org/wiki/General-purpose_operating_system) Linux is also the leading operating system on [servers](https://en.wikipedia.org/wiki/Server_(computing)) and other [big iron](https://en.wikipedia.org/wiki/Big_iron) systems such as [mainframe computers](https://en.wikipedia.org/wiki/Mainframe_computer), and is used on 99.6% of the [TOP500](https://en.wikipedia.org/wiki/TOP500) [super computers](https://en.wikipedia.org/wiki/Supercomputer). It is used by around 2.3% of [desktop computers](https://en.wikipedia.org/wiki/Desktop_computer). The [Chromebook](https://en.wikipedia.org/wiki/Chromebook), which runs the Linux kernel-based [Chrome OS](https://en.wikipedia.org/wiki/Chrome_OS), dominates the US [K–12](https://en.wikipedia.org/wiki/K%E2%80%9312) education market and represents nearly 20% of the sub-$300 [notebook](https://en.wikipedia.org/wiki/Laptop) sales in the US. Linux also runs on [embedded systems](https://en.wikipedia.org/wiki/Embedded_system) – devices whose [operating system](https://en.wikipedia.org/wiki/Operating_system) is typically built into the [firmware](https://en.wikipedia.org/wiki/Firmware) and is highly tailored to the system. This includes TiVo and similar [DVR](https://en.wikipedia.org/wiki/Digital_video_recorder) devices, network [routers](https://en.wikipedia.org/wiki/Router_(computing)), facility automation controls, televisions, video and [smart watches](https://en.wikipedia.org/wiki/Smartwatch).Many smart phones and [tablet computers](https://en.wikipedia.org/wiki/Tablet_computer) run Android and other Linux derivatives.

The development of Linux is one of the most prominent examples of free and open-source [software](https://en.wikipedia.org/wiki/Software) collaboration. The underlying source may be used, modified and distributed‍—‌commercially or non-commercially‍—‌by anyone under the terms of its respective licenses, such as the [GNU General Public License](https://en.wikipedia.org/wiki/GNU_General_Public_License). Typically, Linux is [packaged](https://en.wikipedia.org/wiki/Package_management_system) in a form known as a [Linux distribution](https://en.wikipedia.org/wiki/Linux_distribution)  for both desktop and server use. Some of the most popular mainstream Linux distributions are Linux, CentOS, [Delian](https://en.wikipedia.org/wiki/Debian), [Fedora](https://en.wikipedia.org/wiki/Fedora_(operating_system)), [Gentoo Linux](https://en.wikipedia.org/wiki/Gentoo_Linux), [Linux Mint](https://en.wikipedia.org/wiki/Linux_Mint), [Mageia](https://en.wikipedia.org/wiki/Mageia), [open SUSE](https://en.wikipedia.org/wiki/OpenSUSE) and [Ubuntu](https://en.wikipedia.org/wiki/Ubuntu_(operating_system)), together with commercial distributions such as [Red Hat Enterprise Linux](https://en.wikipedia.org/wiki/Red_Hat_Enterprise_Linux) and [SUSE Linux Enterprise Server](https://en.wikipedia.org/wiki/SUSE_Linux_Enterprise_Server). Distributions include the Linux kernel, supporting [utilities](https://en.wikipedia.org/wiki/System_software) and [libraries](https://en.wikipedia.org/wiki/Library_(computer_science)), many of which are provided by the [GNU Project](https://en.wikipedia.org/wiki/GNU_Project), and usually a large amount of application software to fulfil the distribution's intended use.

* Architecture of Linux

### **Components of Linux kernel**

Linux is based on monolithic kernel. It is able to perform monolithic multitasking in user as well as kernel mode. It is also able to support visual memory. it also provides the facility of shared libraries. It is capable of providing on demand loading. It is also able to do better memory management and threading. It supports shared copy on write executables and inter process communication. It is the architecture of Linux that users have adopted it successfully. Linux is able to perform multi-tasking in a way that it is translucent for the user processes. It seems at times that it is the only process running on the system memory by using main memory and some other useful hardware resources. There are five basic subsystems of Kernel which are process scheduler, memory manager, virtual file system, network interface and Inter process communication. The process scheduler allows and controls the process access to the central processing unit. Memory manager is there to guide the multi processes to make use of main memory in a secure manner. Virtual file system is responsible in making an abridgement of the details of the various hardware devices by in order to present the common file interface to possibly every device. Network interface is the one responsible for providing access to networking protocols and hardware. Inter process communication is complex task. As it is the process of handling variable mechanisms in order to support process to process communication on a one Linux system.

### **The Kernel Software of Linux OS**

The Linux kernel is efficient software. It is capable of performing multitasking. It contains virtual memory, shared libraries, demand loading and memory manager. It can also share copies on write executables. It is also able to do proper memory management and TCP/IP networking. Linux has a monolithic kernel. The kernel extensions and device drivers typically operate in a ring0.  This helps in the full access to the hardware, though some run into user space. Unlike standard monolithic kernels, it is easy to configure as modules loaded and unloaded while the system is running. The monolithic kernel also allows the preemption of drivers. Preemption is also considered for resolving latency, improving the responsiveness of the system and makes the Linux operating system more suitable for the real time applications.

### **File System of Linux**

The file system of Linux is based on single root directory or sub directories. Sub directories are usually used as mount points, where it is possible to combine various network files. The hardware commands are also incorporated into the file hierarchy. Device driver interface to the end user is also a part of this device driver. The information is processed here and is largely mapped to the process directory. The UNIX security system is designed into the architecture this protects the entry of unauthorized or raw hardware to have direct access to the system. Moreover, an internal security system provides individual access to three levels. these three levels include user only, world access, and group membership. Furthermore, every category has executables, read and write options that can be set in any desired combination. It has additional directories for users.

## System Call Execution

The system calls handler gains control when a user program starts a system call. The system call handler changes the protection domain from the caller protection domain, user, to the system call protection domain, kernel, and switches to a protected stack.

The system call handler then calls the function supporting the system call. The loader maintains a table of the currently defined system calls for this purpose. The system calls run within the calling process, but with more privilege than the calling process. This is because the protection domain has changed from user to kernel. The system call function returns to the system call handler when it has performed its operation. The system call handler then restores the state of the process and returns to the user program.

There are two major protection domains in the operating system: the user mode protection domain and the kernel mode protection domain**.**

### Actions of the System Call Handler

When a call is made in user mode that starts a system call, the system call handler is invoked. This system call handler switches the protection domain from user to kernel and performs the following steps:

1. Sets privileged access to the process private address region.
2. Sets privileged access to the kernel address regions.
3. Sets the u t \_error field in the **uthread** structure to 0.
4. Switches to the kernel stack.
5. Starts the specified kernel function (the target of the system call).

On return from the specified kernel function, the system call handler performs the following steps before returning to the caller:

1. Switches back to the user stack.
2. Updates the thread-specific **err no** variable if the u t \_error field is not equal to 0.
3. Clears the privileged access to the kernel address regions.
4. Clears the privileged access to the process private region.
5. Performs signal processing if a signal is pending.

* Command Structure

### Command Structure

Linux commands share the common form

The command identifies the command you want Linux to execute. The name of a Linux command almost always consists of lowercase letters and digits. Remember that, unlike Microsoft Windows, Linux is case sensitive; be sure to type each character of a command in the proper case.

Most commands let you specify options or arguments. However, in any given case, you may not need to do so. For example, typing the w command without options and arguments causes Linux to display a list of current users.

Options modify the way that a command works. Most options consist of a single letter, prefixed by a dash. Often, you can specify more than one option; when you do so, you separate each option with a space or tab. For example, the -h option of the w command causes the output of the command to omit the header lines that give the time and the names of the fields. Typing:

root@desktop:/root#

**w -h**

prints a list of users without the header lines.

Arguments specify filenames or other targets that direct the action of the command. For example, the w command lets you specify a userid as an argument, which causes the command to list only logins that pertain to the specified userid. Typing:

root@desktop:/root#

**w root**

prints a list of current logins by the root user. Some commands let you specify a series of arguments; you must separate each argument with a space or tab.

### 4.2.2 Getting Help

Because Linux provides so many commands and because Linux commands provide so many possible options, you can't expect to recall all of them. To help you, Linux provides the man command and the apropos command, which let you access a help database that describes each command and its options.

#### 4.2.2.1 Using man

Each Linux command is described by a special file called a manual page. The manual pages are stored in a group of subdirectories comprising a help database. To access this database, you use the man command, which resembles the MS-DOS help command. For example, to get help on using the w command, type:

root@desktop:/root#

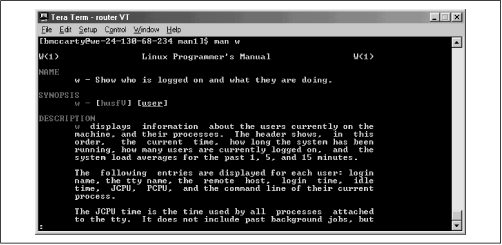
**man w**

[Figure](http://www.oreilly.com/openbook/debian/book/ch04_02.html#CH04-87654)  shows the resulting output, which the command displays one page at a time. Notice the colon prompt, which appears at the bottom left of the screen. To page forward, press the **Space** key; to page backward, press the **b** key. To exit the man program, press the **q** key.

| Manual Page Sections | |
| --- | --- |
| **Section** | **Description** |
| 1 | Executable programs and shell commands |
| 2 | System calls (provided by the kernel) |
| 3 | Library calls (provided by system libraries) |
| 4 | Special files (for example, device files) |
| 5 | File formats and conventions |
| 6 | Games |
| 7 | Macro packages and conventions |
| 8 | System administration commands |
| 9 | Non-standard kernel routines |

The manual pages are organized according to a common format. At the beginning of a manual page, you'll find the name of the page and the section of the manual page database from which the page comes, shown in parentheses. For example, the figure shows the manual page named w, which comes from section 1 of the manual page database. [Table](http://www.oreilly.com/openbook/debian/book/ch04_02.html#CH04-74640)  describes the sections of the manual page database; most sections are primarily of interest to programmers. As a user and administrator, you'll be interested primarily in sections 1 and 8.

#### A typical man page



Next in the output comes the name and brief description of the command. Then comes a synopsis of the command, which shows the options and arguments that you can specify. Brackets enclose parts of a command that you can choose to include or omit. Next comes a detailed description of the operation of the command, followed by a description of its options.

As you're learning your way around Linux, you may find it convenient to reserve a virtual console for running the man command. That way, you can enter commands in a separate virtual console, switching between consoles to refresh your recollection of the options and arguments of commands as you type them.

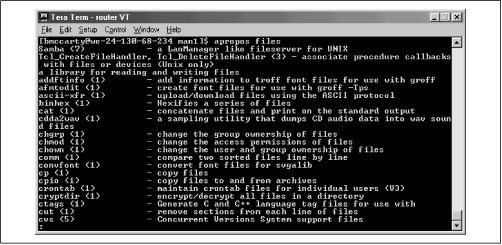
#### Using apropos

The man command searches the manual pages and displays detailed information about a specified command. The apropos command also searches the manual pages; however, it displays summary information about manual pages that contain a specified keyword. (The search is limited to the short description that appears at the beginning of each manual page). For example, typing the command:

root@desktop:/root#

**apropos files**

displays a list of manual pages that contain the word files, as shown in Figure



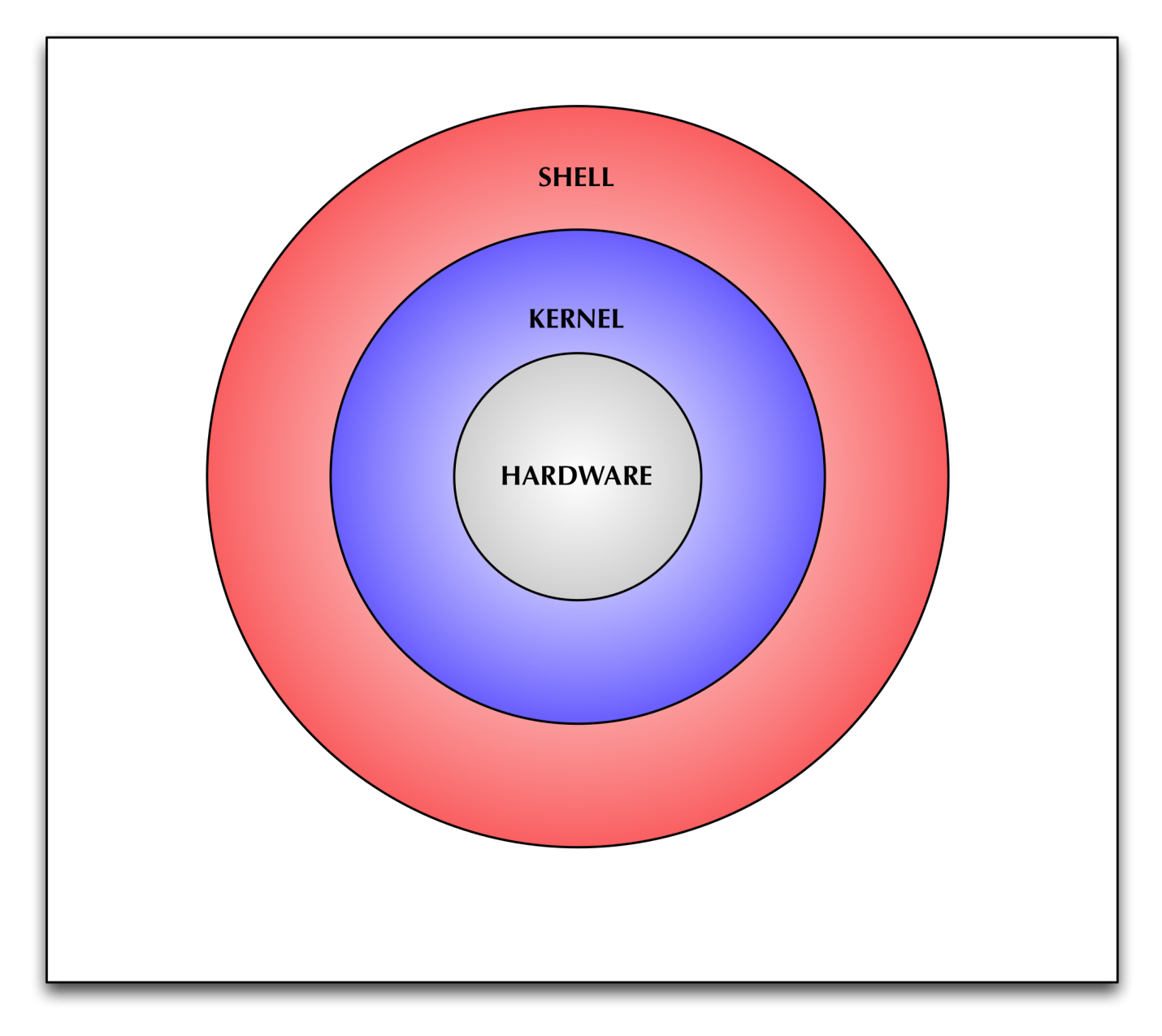
The apropos command is useful when you don't recall the name of a Linux command. By typing a related keyword, you can obtain a list of commands and search the list for the command you need.

**Assignment 2**

Concept of Shell

The shell provides you with an interface to the UNIX system. It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.

A shell is an environment in which we can run our commands, programs, and shell scripts. There are different flavors of shells, just as there are different flavors of operating systems. Each flavor of shell has its own set of recognized commands and functions.





Shell Structure

Shell Types: -

In UNIX there are two major types of shells:

1. The Bourne shell. If you are using a Bourne-type shell, the default prompt is the $ character.
2. The C shell. If you are using a C-type shell, the default prompt is the % character.
3. There are again various subcategories for Bourne Shell which are listed as follows:

* Bourne shell (sh)
* Korn shell (ksh)
* Bourne Again shell (bash)
* POSIX shell (sh)

The different C-type shells follow:

* C shell (csh)
* TENEX/TOPS C shell (tcsh)

The original UNIX shell was written in the mid-1970s by Stephen R. Bourne while he was at AT&T Bell Labs in New Jersey.

The Bourne shell was the first shell to appear on UNIX systems, thus it is referred to as "the shell".

The Bourne shell is usually installed as /bin/sh on most versions of UNIX. For this reason, it is the shell of choice for writing scripts to use on several different versions of UNIX.

In this tutorial, we are going to cover most of the Shell concepts based on Borne Shell.

**Is Command**

**ls** List information about the FILEs (the current directory by default). Sort entries alphabetically if none of -cftuvSUX nor --sort is specified. Mandatory arguments to long options are mandatory for short options too.

Syntax is: - ls [OPTION]... [FILE]...

Some examples of options are given below; -

| Option | Description |
| --- | --- |
| -a | Displays all files. |
| -b | Displays nonprinting characters in octal. |
| -c | Displays files by file timestamp. |
| -C | Displays files in a columnar format (default) |
| -d | Displays only directories. |
| -f | Interprets each *name* as a directory, not a file. |
| -F | Flags filenames. |

CAT Command

The cat command is considered as one of the most frequently used commands on Linux or UNIX like operating systems.

It can be used for the following purposes under UNIX or Linux:

* Display text files on screen.
* Copy text files.
* Combine text files.
* Create new text files.
* **cat command Syntax**
* The syntax is as follows:

|  |
| --- |
| **cat** filename  **cat** options filename  **cat** file1 file2  **cat** file1 file2 **>** newcombinedfile |

Displaying The Contents of Files

To read or read the contents of files, enter:  
$ cat /etc./passwd

Copy File

The cat command can also be used to create a new file and transfer to it the data from an existing file. To make copy of  
$ cat oldfile.txt > newfile.txt

Create a File

You can use cat command for file creation. To create a file called foo.txt, enter:  
$ cat > foo.txt

Concatenate files

Concatenation means putting multiple file contents together. The original file or files are not modified or deleted. In this example, cat will concatenate copies of the contents of the three files /etc/hosts, /etc/resolv.conf, and /etc

# [**Pass the output of previous command to next as an argument**](https://unix.stackexchange.com/questions/108782/pass-the-output-of-previous-command-to-next-as-an-argument)

 a mechanism for sending data from one program to another. It's called piping and the operator we use is (|) (found above the backslash (\) key on most keyboards). What this operator does is feed the output from the program on the left as input to the program on the right. In the example below we will list only the first 3 files in the directory.

1. **ls**
2. barry.txt bob example.png first file foo1 myoutput video.mpeg
3. **ls | head -3**
4. barry.txt
5. bob
6. example.png
7. user@bash

We may pipe as many programs together as we like. In the below example we have then piped the output to tail so as to get only the third file.

1. **ls | head -3 | tail -1**
2. example.png
3. user@bash

# Linux Logout user / Logoff User Commands

a) **pkill command** – Kill processes by name.

b) **kill command** – terminate or signal a process.

c) **logout command** – Logout of a login shell. This command can be used by normal users to end their own session.

Locating Commands

The **locate command** is often the simplest and quickest way to find the locations of files and directories on Linux and other Unix-like operating systems.

|  |  |
| --- | --- |
|  | The *locate* [command](http://www.linfo.org/command.html) is often the simplest and quickest way to find the locations of files and directories on [Linux](http://www.linfo.org/linuxdef.html) and other [Unix-like](http://www.linfo.org/unix-like.html) [operating systems](http://www.linfo.org/operating_systems_list.html).  The basic syntax for locate is:  locate [options] name(s)  PATH Variable  **PATH** is an environment **variable** on Unix-like operating systems, DOS, OS/2, and Microsoft Windows, specifying a set of directories where executable programs are located. In general, each executing process or user session has its own **PATH** setting.  SHELL variable  Unix - Using **Shell Variables**. Advertisements. ... 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. A **variable** is nothing more than a pointer to the actual data. |

Combining Commands

Chaining of **Linux commands** means, **combining** several **commands** and make them execute based upon the behaviour of operator used in between them. Chaining of **commands in Linux**, is something like you are writing short shell scripts at the shell itself, and executing them from the terminal

## The Semicolon (;) Operator

ls; pwd; whoami

## The Logical AND Operator (&&)

* mkdir MyFolder && cd MyFolder

## The Logical OR Operator (||)

* [ -d ~/MyFolder] || mkdir ~/MyFolder

## Echo and Echo-6

## The echo Command. echo is a built-in command in the bash and C shells that writes its arguments to standard output. A shell is a program that provides the command line (i.e., the all-text display user interface) on Linux and other Unix-like operating systems

|  |  |
| --- | --- |
|  | *echo* is a built-in [*command*](http://www.linfo.org/command.html) in the *bash* and *C* [*shells*](http://www.linfo.org/shell.html) that writes its [*arguments*](http://www.linfo.org/argument.html) to [*standard output*](http://www.linfo.org/standard_output.html).  A shell is a [program](http://www.linfo.org/program.html) that provides the [command line](http://www.linfo.org/command_line.html) (i.e., the all-text display user [interface](http://www.linfo.org/interface.html)) on [Linux](http://www.linfo.org/linuxdef.html) and other [Unix-like](http://www.linfo.org/unix-like.html) [operating systems](http://www.linfo.org/operating_systems_list.html). It also executes (i.e., runs) commands that are typed into it and displays the results. bash is the default shell on Linux.  A command is an instruction telling a computer to do something. An argument is input data for a command. Standard output is the display screen by default, but it can be [redirected](http://www.linfo.org/redirection.html) to a file, printer, etc.  The syntax for echo is  echo [option(s)] [string(s)] |

echo by itself displays a line of text. It will take anything within the following "..." two quotation marks, literally, and just print out as it is. However, with echo -e you're making echo to enable interpret backslash escapes. So with this in mind here are some examples

INPUT: echo "abc\n def \nghi"

OUTPUT:abc\n def \nghi

INPUT: echo -e "abc\n def \nghi"

OUTPUT:abc

def

ghi

Note: \n is new line, ie a carriage return. If you want to know what other sequences are recognized by echo -e type in man echo to your terminal.

Cal and date command

**cal** - To display a calendar. A single parameter specifies the 4-digit year (1 - 9999) to be displayed. Two parameters denote the Month (1 - 12) and Year (1 - 9999). If arguments are not specified, the current month is displayed

## OPTIONS

|  |  |
| --- | --- |
| **Tag** | **Description** |
| -m | Display monday as the first day of the week. |
| -j | Display julian dates (days one-based, numbered from January 1). |
| -y | Display a calendar for the current year. |

## EXAMPLES

To display current month's calendar

$ cal

April 2016

Su Mo Tu We Th Fr Sa

1 2

3 4 5 6 7 8 9

10 11 12 13 14 15 16

17 18 19 20 21 22 23

24 25 26 27 28 29 30

## The date command is used to print out, or change the value of, the system's time and date information. date syntax

date [*OPTION*]... [+*FORMAT*]

date [-u|--utc|--universal] [*MMDDhhmm*[[*CC*]*YY*] [. *ss*]]

## Options

|  |  |
| --- | --- |
| **-d**, **--date=***STRING* | Display time described by [string](https://www.computerhope.com/jargon/s/string.htm) *STRING*, as opposed to the default, which is '**now**'. |
| **-f**, **--file=***DATEFILE* | Like **--date**, but processed once for each line of file *DATEFILE*. |
| **-I**[*TIMESPEC*],**--iso-8601**[**=***TIMESPEC*] | Output date/time in [ISO](https://www.computerhope.com/jargon/i/iso.htm) 8601 format. For values of *TIMESPEC*, use '**date**' for date only (the default), '**hours**', '**minutes**', '**seconds**', or '**ns**' for date and time to the indicated precision. |
| **-r**, **--reference=***FILE* | Display the last modification time of file *FILE*. |
| **-R**, **--rfc-2822** | Output date and time in [RFC](https://www.computerhope.com/jargon/r/rfc.htm) 2822 format. Example: **Mon, 07 Aug 2006 12:34:56 -0600** |
| **--rfc-3339=***TIMESPEC* | Output date and time in RFC 3339 format. *TIMESPEC* can be set to '**date**', '**seconds**', or '**ns**' for date and time to the indicated precision. Date and time components are separated by a single space, for example:**2006-08-07 12:34:56-06:00** |
| **-s**, **--set=***STRING* | Set time described by string *STRING*. |
| **-u**, **--utc**, **--universal** | Print or set Coordinated Universal Time. |
| **--help** | Display a help message and exit. |
| **--version** | Display version information and exit. |

## Date Format

*FORMAT* is a sequence of characters which specifies how output will appear. It comprises some combination of the following sequences:

|  |  |
| --- | --- |
| **%%** | A literal percent sign ("**%**"). |
| **%a** | The abbreviated weekday name (e.g., **Sun**). |
| **%A** | The full weekday name (e.g., **Sunday**). |
| **%b** | The abbreviated month name (e.g., **Jan**). |
| **%B** | Locale's full month name (e.g., **January**). |
| **%c** | The date and time (e.g., **Thu Mar 3 23:05:25 2005**). |
| **%C** | The current century; like **%Y**, except omit last two digits (e.g., **20**). |
| **%d** | Day of month (e.g., **01**). |
| **%D** | Date; same as **%m/%d/%y**. |
| **%e** | Day of month, space padded; same as **%\_d**. |
| **%F** | Full date; same as **%Y-%m-%d**. |
| **%g** | Last two digits of year of ISO week number (see **%G**). |
| **%G** | Year of ISO week number (see **%V**); normally useful only with **%V**. |
| **%h** | Same as **%b**. |
| **%H** | Hour (**00**..**23**). |
| **%I** | Hour (**01**..**12**). |
| **%j** | Day of year (**001**..**366**). |
| **%k** | Hour, space padded ( **0**..**23**); same as **%\_H**. |
| **%l** | Hour, space padded ( **1**..**12**); same as **%\_I**. |
| **%m** | Month (**01**..**12**). |
| **%M** | Minute (**00**..**59**). |
| **%n** | A [newline](https://www.computerhope.com/jargon/n/newline.htm). |
| **%N** | [Nanoseconds](https://www.computerhope.com/jargon/n/nanosec.htm) (**000000000**..**999999999**). |
| **%p** | Locale's equivalent of either **AM** or **PM**; blank if not known. |
| **%P** | Like **%p**, but lower case. |
| **%r** | Locale's 12-hour clock time (e.g., **11:11:04 PM**). |
| **%R** | 24-hour hour and minute; same as **%H:%M**. |
| **%s** | Seconds since 1970-01-01 00:00:00 UTC. |
| **%S** | [Second](https://www.computerhope.com/jargon/s/second.htm) (**00**..**60**). |
| **%t** | A [tab](https://www.computerhope.com/jargon/t/tab.htm). |
| **%T** | Time; same as **%H:%M:%S**. |
| **%u** | Day of week (**1**..**7**); 1 is **Monday**. |
| **%U** | Week number of year, with Sunday as first day of week (**00**..**53**). |
| **%V** | ISO week number, with Monday as first day of week (**01**..**53**). |
| **%w** | Day of week (**0**..**6**); 0 is **Sunday**. |
| **%W** | Week number of year, with Monday as first day of week (**00**..**53**). |
| **%x** | Locale's date representation (e.g., **12/31/99**). |
| **%X** | Locale's time representation (e.g., **23:13:48**). |
| **%y** | Last two digits of year (**00**..**99**). |
| **%Y** | Year. |
| **%z** | +hhmm numeric time zone (e.g., **-0400**). |
| **%:z** | +hh:mm numeric time zone (e.g., **-04:00**). |
| **%::z** | +hh:mm:ss numeric time zone (e.g., **-04:00:00**). |
| **%:::z** | Numeric time zone with "**:**" to necessary precision (e.g., **-04**, **+05:30**). |
| **%Z** | Alphabetic time zone abbreviation (e.g., EDT). |

By default, date pads numeric fields with zeroes. The following optional flags may follow '**%**':

|  |  |
| --- | --- |
| **-** | ([Hyphen](https://www.computerhope.com/jargon/h/hyphen.htm)) do not pad the field. |
| **\_** | Pad with [spaces](https://www.computerhope.com/jargon/s/space.htm). |
| **0** | Pad with zeros. |
| **^** | Use upper case if possible. |
| **#** | Use opposite case if possible. |

## Man and Help Command

## man is the system's manual viewer; it can be used to display manual pages, scroll up and down, search for occurrences of specific text, and other useful functions.

## man syntax

man -k [*apropos options*] *regexp* ...

## HELP

## Display information about builtin commands.

Displays brief summaries of [shell](https://www.computerhope.com/jargon/s/shell.htm) builtin commands. If *PATTERN* is specified, gives detailed help on all commands matching *PATTERN*, otherwise the list of help topics is printed.

## help syntax

help [-dms] [*PATTERN* ...]

# **Escape Characters**

Certain characters are significant to the shell; we have seen, for example, that the use of double quotes (") characters affect how spaces and TAB characters are treated, for example:

$ echo Hello World

Hello World

$ echo "Hello World"

Hello World

So how do we display: **Hello    "World”?**

$ echo "Hello "World""

The first and last " characters wrap the whole lot into one parameter passed to **echo** so that the spacing between the two words is kept as is. But the code:

$ echo "Hello " World ""

would be interpreted as three parameters:

1. "Hello   "
2. World
3. ""

So the output would be

Hello World

Note that we lose the quotes entirely. This is because the first and second quotes mark off the Hello and following spaces; the second argument is an unquoted "World" and the third argument is the empty string; "".

Thanks to Patrick for pointing out that this:

$ echo "Hello "World""

is actually only one parameter (no spaces between the quoted parameters), and that you can test this by replacing the **echo** command with (for example) **ls**.

Most characters (**\***, **'**, etc) are not interpreted (ie, they are taken literally) by means of placing them in double quotes (""). They are taken as is and passed on to the command being called. An example using the asterisk (\*) goes:

$ echo \*

case.shtml escape.shtml first.shtml

functions.shtml hints.shtml index.shtml

ip-primer.txt raid1+0.txt

$ echo \*txt

ip-primer.txt raid1+0.txt

$ echo "\*"

\*

$ echo "\*txt"

\*txt

In the first example, \* is expanded to mean all files in the current directory.  
In the second example, \*txt means all files ending in **txt**.  
In the third, we put the \* in double quotes, and it is interpreted literally.  
In the fourth example, the same applies, but we have appended **txt** to the string.

However, **"**, **$**, **`**, and **\** are still interpreted by the shell, even when they're in double quotes.   
The backslash (\) character is used to mark these special characters so that they are not interpreted by the shell, but passed on to the command being run (for example, **echo**).   
So to output the string: (Assuming that the value of **$X** is 5):

A quote is ", backslash is \, backtick is `.

A few spaces are and dollar is $. $X is 5.

we would have to write:

$ echo "A quote is \", backslash is \\, backtick is \`."

A quote is ", backslash is \, backtick is `.

$ echo "A few spaces are; dollar is \$. \$X is ${X}."

A few spaces are; dollar is $. $X is 5.

We have seen why the " is special for preserving spacing. Dollar (**$**) is special because it marks a variable, so **$X** is replaced by the shell with the contents of the variable **X**. Backslash (**\**) is special because it is itself used to mark other characters off; we need the following options for a complete shell:

$ echo "This is \\ a backslash"

This is \ a backslash

$ echo "This is \" a quote and this is \\ a backslash"

This is " a quote and this is \ a backslash

So backslash itself must be escaped to show that it is to be taken literally. The other special character, the backtick, is discussed later in Chapter 12, [External Programs](https://www.shellscript.sh/external.html).

## Printf

## printf prints a *formatted string* to the standard output. Its roots are in the [C programming language](https://www.computerhope.com/jargon/c/c.htm), which uses a [function](https://www.computerhope.com/jargon/f/function.htm) by the same name. It is a handy way to produce precisely-formatted output from numerical or textual arguments.

## printf syntax

printf *FORMAT* [*ARGUMENT*]...

printf *OPTION*

## Options

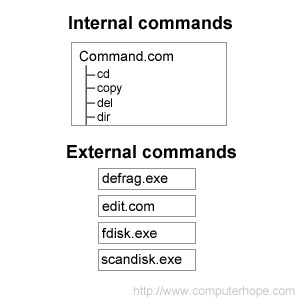
|  |  |
| --- | --- |
| *FORMAT* | *FORMAT* controls the output, and defines the way that the *ARGUMENT*s will be expressed in the output. See the [Format](https://www.computerhope.com/unix/uprintf.htm#Format) section, below. |
| *ARGUMENT* | Each *ARGUMENT* will be inserted into the formatted output according to the definition of*FORMAT*. |
| **--help** | Display a help message, and exit. |
| **--version** | Display version information, and exit. |

## 

Assignment 3

**Internal** **commands**

**Internal** **commands** are functions that are built into the **command** interpreter, **External** **commands** are those not included in the interpreter, and are instead invoked by calling an **external** binary. Whether or not a particular **command** is **internal** or **external** varies by system.



Listing of internal commands

Below are examples of internal commands in MS-DOS and the Windows command line currently listed in the Computer Hope database.

|  |  |  |  |
| --- | --- | --- | --- |
| [Assoc](https://www.computerhope.com/assoc.htm) [Atmadm](https://www.computerhope.com/atmadm.htm) [Break](https://www.computerhope.com/breakhlp.htm) [Call](https://www.computerhope.com/call.htm) [CD](https://www.computerhope.com/cdhlp.htm) [Chdir](https://www.computerhope.com/chdirhlp.htm) [Cls](https://www.computerhope.com/clshlp.htm) [Color](https://www.computerhope.com/color.htm) [Copy](https://www.computerhope.com/copyhlp.htm) [Ctty](https://www.computerhope.com/cttyhlp.htm) [Date](https://www.computerhope.com/datehlp.htm) [Del](https://www.computerhope.com/delhlp.htm) [Dir](https://www.computerhope.com/dirhlp.htm) [Drivparm](https://www.computerhope.com/drivparm.htm) [Echo](https://www.computerhope.com/echohlp.htm) [Endlocal](https://www.computerhope.com/endlocal.htm) [Erase](https://www.computerhope.com/erasehlp.htm) [Exit](https://www.computerhope.com/exithlp.htm) | [For](https://www.computerhope.com/forhlp.htm) [Goto](https://www.computerhope.com/goto.htm) [If](https://www.computerhope.com/if.htm) [LH Loadhigh](https://www.computerhope.com/lhhlp.htm) [Lock](https://www.computerhope.com/lockhlp.htm) [Md Mkdir](https://www.computerhope.com/mdhlp.htm) [Mklink](https://www.computerhope.com/mklink.htm) [Move](https://www.computerhope.com/movehlp.htm) [Path](https://www.computerhope.com/pathhlp.htm) [Pause](https://www.computerhope.com/pausehlp.htm) [Popd](https://www.computerhope.com/popd.htm) [Prompt](https://www.computerhope.com/prompthl.htm) [Pushd](https://www.computerhope.com/pushd.htm) | [Rd](https://www.computerhope.com/rmdirhlp.htm) [Ren Rename](https://www.computerhope.com/renamehl.htm) [Rmdir](https://www.computerhope.com/rmdirhlp.htm) [Set](https://www.computerhope.com/sethlp.htm) [Setlocal](https://www.computerhope.com/setlocal.htm) [Shift](https://www.computerhope.com/shift.htm) [Start](https://www.computerhope.com/starthlp.htm) [Switches](https://www.computerhope.com/switches.htm) [Time](https://www.computerhope.com/timehlp.htm) [Title](https://www.computerhope.com/titlehlp.htm) [Type](https://www.computerhope.com/typehlp.htm) | [Unlock](https://www.computerhope.com/lockhlp.htm) [Ver](https://www.computerhope.com/verhlp.htm) [Verify](https://www.computerhope.com/verifyhl.htm) [Vol](https://www.computerhope.com/volhlp.htm) |

Where are the internal command files stored?

All of the internal commands are part of command.com or cmd.exe (depending on your version of MS-DOS or Windows) and are not separate files on the hard drive.

How do you run an external command?

As long as you can open a command line you can run any of the Internal commands included with your version of MS-DOS or Microsoft Windows.

Commands:-

* Stty

stty changes and prints [terminal](https://www.computerhope.com/jargon/t/terminal.htm) line settings.

Description

stty displays or changes the characteristics of the terminal.

stty syntax

stty [-F *DEVICE* | --file=*DEVICE*] [*SETTING*]...

stty [-F *DEVICE* | --file=*DEVICE*] [-a|--all]

stty [-F *DEVICE* | --file=*DEVICE*] [-g|--save]

Options

|  |  |
| --- | --- |
| -a, --all | Print all current settings in human-readable form. |
| -g, --save | Print all current settings in a stty-readable form. |
| -F, --file=*DEVICE* | Open and use the specified *DEVICE* instead of [stdin.](https://www.computerhope.com/jargon/s/stdin.htm) |
| --help | Display a help message, and exit. |
| --version | Output version information, and exit. |

## stty examples

stty sane

Reset all terminal settings to "sane" values; this has the effect of "fixing" the terminal when another program alters the terminal settings to an unusable condition.

* Who

Displays who is [logged on](https://www.computerhope.com/jargon/s/signon.htm) to the [system](https://www.computerhope.com/jargon/s/system.htm).

Description

The who command prints information about all users who are currently logged in.

who Syntax

who [ *OPTION*] ... [ *FILE*] [ am i]

Options

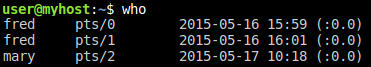
|  |  |
| --- | --- |
| -a, --all | Same as using the options -b -d --login -p -r -t -T -u. |
| -b, --boot | Display the time of the last system [boot](https://www.computerhope.com/jargon/b/boot.htm). |
| -d, --dead | Display dead [processes](https://www.computerhope.com/jargon/p/process.htm). |
| -H, --heading | Print a line of column headings. |
| --ips | Print [IP](https://www.computerhope.com/jargon/i/ip.htm) addresses instead of [hostnames](https://www.computerhope.com/jargon/h/hostname.htm). With --lookup, canonicalizes based on stored IP, if available, rather than stored hostname. |

If the arguments "am i" are specified, who assumes the -m option.

## who examples

who

Displays the username, line, and time of all currently logged-in sessions. For example:



* Uname

Print information about the current system.

Description

Print certain [system](https://www.computerhope.com/jargon/s/system.htm) information. If no *OPTION* is specified, uname assumes the -s option.

uname syntax

uname [*OPTION*]...

Options

|  |  |
| --- | --- |
| -a, --all | Prints all information, omitting -p and -i if the information is unknown. |

If -a (--all) is specified, the information is printed in the following order of individual options:

|  |  |
| --- | --- |
| -s, --kernel-name | Print the [kernel](https://www.computerhope.com/jargon/k/kernel.htm) name. |
| -n, --nodename | Print the [network](https://www.computerhope.com/jargon/n/network.htm) [node](https://www.computerhope.com/jargon/h/hostcomp.htm) [hostname](https://www.computerhope.com/jargon/h/hostname.htm). |
| -r, --kernel-release | Print the [kernel](https://www.computerhope.com/jargon/k/kernel.htm) [release](https://www.computerhope.com/jargon/r/release.htm). |
| -v, --kernel-version | Print the kernel version. |
| -m, --machine | Print the machine [hardware](https://www.computerhope.com/jargon/h/hardware.htm) name. |
| -p, --processor | Print the [processor](https://www.computerhope.com/jargon/c/cpu.htm) type, or "unknown". |
| -i, --hardware-platform | Print the hardware platform, or "unknown". |
| -o, --operating-system | Print the [operating system](https://www.computerhope.com/os.htm). |
| --help | Display a help message, and exit. |
| --version | Display version information, and exit. |

uname examples

uname -a

Displays system information. Output will resemble the following:

SunOS hope 5.7 Generic\_106541-08 sun4m sparc SUNW, SPARCstation-10

[Print](https://www.computerhope.com/jargon/p/print.htm) the [file name](https://www.computerhope.com/jargon/f/filename.htm) of the [terminal](https://www.computerhope.com/jargon/t/terminal.htm) connected to [standard input](https://www.computerhope.com/jargon/s/stdin.htm).

* tty

tty syntax

tty [*OPTION*]...

Options

|  |  |
| --- | --- |
| -s, --silent, --quiet | Print nothing, only return an exit status. |
| --help | display this help and exit. |
| --version | output version information and exit. |

tty examples

tty

Running tty by itself will display the current tty session as shown below:

/dev/pts/0

* Types of files

1. Directory files
2. Special files(This category is having 5 sub types in it.)

So in practical we have total 7 types (1+1+5) of files in Linux/Unix. And in Solaris we have 8 types. And you can see the file type indication at leftmost part of “ls -l” command.

Here are those files type.

1. **Regular file (-)**
2. **Directory files(d)**

**Special files**

1. **Block file(b)**
2. **Character device file(c)**
3. **Named pipe file or just a pipe file(p)**
4. **Symbolic link file(l)**
5. **Socket file(s)**

For your information there is one more file type called door file(D) which is present in Sun Solaris as mention earlier. A door is a special file for inter-process communication between a client and server (so total 8 types in Unix machines). We will learn about different types of files as below sequence for every file type.

**Definition and information of the file type**

**How to create particular file type**

**How to list/see particular file type**

## **Regular file type in Linux**

These are the files which are indicated with "-" in ls -l command output at the starting of the line. And these files are.

1. Readable file or

2. A binary file or

3. Image files or

4. Compressed files etc.

**How to create regular files in Linux/Unix?**  
Ans: Use touch/vi command and [redirection operators](http://www.linuxnix.com/linuxunix-redirection-operatorsfile-descriptors-explained-examples/) etc.

**How can we list regular files?**

**ls -l | grep ^-**

Example listing of regular files:

-rw-r--r-- 1 krishna krishna 20986522 2010-01-31 13:48 test.wmv

-rw-r--r-- 1 krishna krishna 173448 2010-01-30 21:20 Transformers-Teaser-Wallpaper-310.jpg

-r-xr-xr-x 1 root root 135168 2009-12-12 19:14 VIDEO\_TS.VOB

-rw-r--r-- 1 krishna krishna 2113536 2009-12-01 13:32 Aditya 365 – Janavule.mp3

-rwxrwxrwx 1 root root 168 2010-02-14 14:12 xyz.sh

## **Directory file type explained in Linux/Unix**

These type of files contains regular files/folders/special files stored on a physical device. And this type of files will be in blue in color with link greater than or equal 2.

**How can we list them in my present working directory?**

**ls -l | grep ^d**

Example listing of directories.

drwxr-xr-x 2 surendra surendra 4096 2010-01-19 18:37 bin

drwxr-xr-x 5 surendra surendra 4096 2010-02-15 18:46 Desktop

drwxr-xr-x 2 surendra surendra 4096 2010-01-18 14:36 Documents

drwxr-xr-x 2 surendra surendra 4096 2010-02-13 17:45 Downloads

# **Linux File systems**

|  |
| --- |
| Contents   1. [Introduction to File systems](https://help.ubuntu.com/community/LinuxFilesystemsExplained#Introduction_to_File_systems) 2. [Journaling](https://help.ubuntu.com/community/LinuxFilesystemsExplained#Journaling) 3. [Table](https://help.ubuntu.com/community/LinuxFilesystemsExplained#Table) 4. [Editing Files](https://help.ubuntu.com/community/LinuxFilesystemsExplained#Editing_Files) 5. [Fragmentation](https://help.ubuntu.com/community/LinuxFilesystemsExplained#Fragmentation) 6. [See Also](https://help.ubuntu.com/community/LinuxFilesystemsExplained#See_Also) 7. [Other Resources](https://help.ubuntu.com/community/LinuxFilesystemsExplained#Other_Resources) 8. [Different File Systems on the same disk](https://help.ubuntu.com/community/LinuxFilesystemsExplained#Different_File_Systems_on_the_same_disk) |

# **Introduction to File systems**

File systems are one of the things any newcomer to linux must become acquainted with. In the world of Microsoft, you never really have to worry about it, the default being NTFS. Linux however, being built on a world of open source and differing opinions, is not limited in this way and so the user should have an understanding of what a file system is, and how it affects the computer.

At the core of a computer, it's all 1s and 0s, but the organization of that data is not quite as simple. A bit is a 1 or a 0, a byte is composed of 8 bits, a kibibyte is 1024 (i.e. 2^10) bytes, a mebibyte is 1024 kibibytes and so on and so forth. All these bits and bytes are permanently stored on a Hard Drive. A hard drive stores all your data, any time you save a file, you're writing thousands of 1s and 0s to a metallic disc, changing the magnetic properties that can later be read as 1 or 0. There is so much data on a hard drive that there has to be some way to organize it, like a library of books and the old card drawers that indexed all of them, without that index, we'd be lost. Libraries, for the most part, use the Dewey Decimal System to organize their books, but there exist other systems to do so, none of which have attained the same fame as Mr. Dewey's invention. File systems are the same way. The ones most users are aware of are the ones Windows uses, the vFat or the NTFS systems, these are the Windows default file systems.

# **Table**

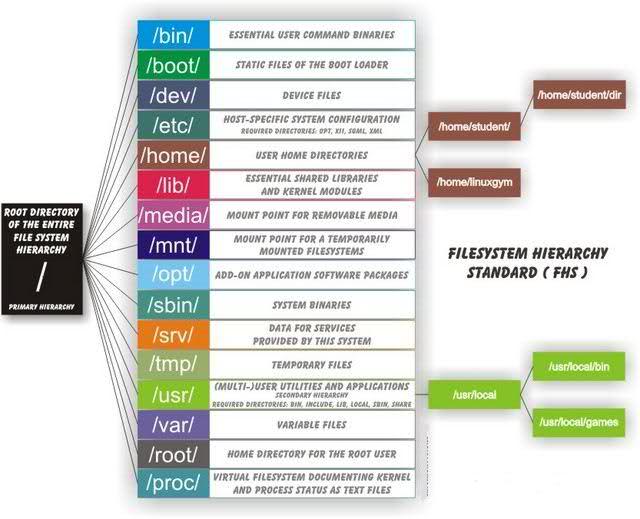
Now below is a very brief comparison of the most common file systems in use with the Linux world.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| File System | Max File Size | Max Partition Size | Journaling | Notes |
| Fat16 | 2 GiB | 2 GiB | No | Legacy |
| Fat32 | 4 GiB | 8 TiB | No | Legacy |
| NTFS | 2 TiB | 256 TiB | Yes | (For Windows Compatibility) NTFS-3g is installed by default in Ubuntu, allowing Read/Write support |
| ext2 | 2 TiB | 32 TiB | No | Legacy |
| ext3 | 2 TiB | 32 TiB | Yes | Standard linux filesystem for many years. Best choice for super-standard installation. |
| ext4 | 16 TiB | 1 EiB | Yes | Modern iteration of ext3. Best choice for new installations where super-standard isn't necessary. |
| reiserFS | 8 TiB | 16 TiB | Yes | No longer well-maintained. |
| JFS | 4PiB | 32PiB | Yes (metadata) | Created by IBM - Not well maintained. |
| XFS | 8 EiB | 8 EiB | Yes (metadata) | Created by SGI. Best choice for a mix of stability and advanced journaling. |
| GiB = Gibibyte (1024 MiB) :: TiB = Tebibyte (1024 GiB) :: PiB = Pebibyte (1024 TiB) :: EiB = Exbibyte (1024 PiB) | | | | |

* Linux Directory Structure

### Linux Directory Structure Diagram

A standard **Linux** distribution follows the directory structure as provided below with Diagram and explanation.



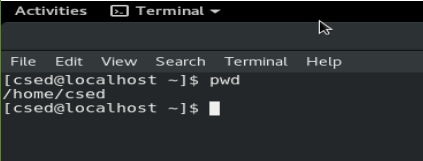
Home Variables

HOME contains the path to the home directory of the current user. This variable can be used by applications to associate configuration files and such like with the user running it.

Commands:-

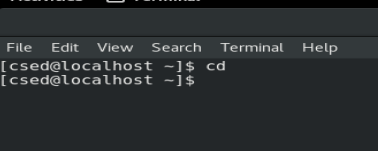
## Pwd command

Pwd is the basic command that can be useful to find out where you are in the linux system (your current directory path), this simple example the step to use Linux pwd command on the bash shell command prompt to print current working directory on Linux fedora core system.



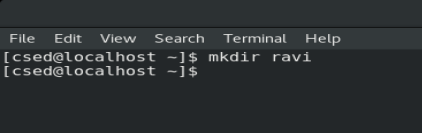
## cd

The **cd** **command**, also known as chdir (change directory), is a command-line OS shell command used to change the current working directory in operating systems such as Unix, DOS, OS/2, AmigaOS (where if a bare path is given, **cd** is implied), Windows, and Linux. It is also available for use in shell scripts and batch files.



## mkdir

The **mkdir** (make directory) **command** in the Unix, DOS, OS/2, and Microsoft Windows operating systems and in the PHP scripting language is used to make a new directory. In DOS, OS/2 and Windows, the command is often abbreviated to md.



* Absolute Path name

An absolute path is defined as the specifying the location of a file or directory from the root directory (/). In other words, we can say absolute path is a complete path from start of actual filesystem from / directory.

#### **Some examples of absolute path:**

**/var/**[**ftp**](http://www.linuxnix.com/ftps-server-configuration/)**/pub**

[**/etc**](http://www.linuxnix.com/linux-directory-structure-explainedetc-folder/)**/samba.smb.conf**

**/boot/grub/grub.conf**

## **Rrelative path**

Relative path is defined as path related to the present working directory(pwd). Suppose I am located in /var/log and I want to change directory to /var/log/kernel. I can use relative path concept to change directory to kernel

changing directory to /var/log/kernel by using relative path concept.

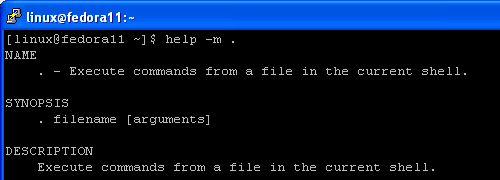
**pwd**

**/var/log**

**cd kernel**

# **Get Linux dot ( . ) command**

  Execute help command and the name of Linux command that we want the help file, the example below we execute help command with -m option to get the help for dot (.) command using the manpage format.

[](http://www.basiclinuxcommand.com/files/builtin_command/dot/dot_b.png)

Dot and Dot Dot (. And..)

"." referes to the current directory. This is important, for example, when you are compiling software. Very often there is a configure script to configure how the software should be built. You would enter the command './configure' from within the source directory of the new software, and the "." indicates to the OS that the particular configure script you want to use is in the current directory.  
  
".." points to the directory directly above working directory you are in at the time. So, if you drop down into your bin directory to edit one of your scripts and then want to go back to the next directory higher, you can type the command 'cd ..' to go there.

Commands: -

* Cat

The **cat** (short for “concatenate “) **command** is one of the most frequently used **command** in **Linux**/Unix like operating systems. **cat** **command** allows us to create single or multiple files, view contain of file, concatenate files and redirect output in terminal or files.

# cat /etc/passwd

root:x:0:0:root:/root:/bin/bash

bin:x:1:1:bin:/bin:/sbin/nologin

narad:x:500:500::/home/narad:/bin/bash

* Cp

The **cp** **command** is quite important and used frequently when we work in the **command** line terminal. Normally, we use the **cp** **command** to backup a file by making extra copy of the original file.

Eg

NAME  
cp - copy files and directories  
  
SYNOPSIS  
cp [OPTION]... [-T] SOURCE DEST  
cp [OPTION]... SOURCE... DIRECTORY  
cp [OPTION]... -t DIRECTORY SOURCE...  
  
DESCRIPTION  
Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.

* Rm

If you run this command as root (e.g. with sudo rm -rf /), Linux actually refuses to run it. You need to pass a special option --no-preserve-root in order to call rm on /. You can still run dangerous commands, e.g. rm -rf ~ will delete all your files.

For instance, if there is a file in your current directory named "**-file.txt**", you can delete it with the command

rm -- -file.txt

Or, you can delete it by referring to it with a [pathname](https://www.computerhope.com/jargon/p/path.htm). For instance, if the file "**-file.txt**" was located in the directory "**/home/chope**", you could delete it using:

rm /home/chope/-file.txt

...or, if **/home/chope** is your [current directory](https://www.computerhope.com/jargon/c/currentd.htm),

rm ./-file.txt

* mv

**mv** renames file *SOURCE* to *DEST*, or moves the *SOURCE* file (or files) to *DIRECTORY*.

## mv syntax

mv [*OPTION*]... [**-T**] *SOURCE* *DEST*

mv [*OPTION*]... *SOURCE*... *DIRECTORY*

* **wc**

it prints newline, word, and byte counts for each *FILE*, and a total if more than one *FILE* is specified. With no *FILE*, or when *FILE* is a dash ("**-**"), **wc** operates on [standard input](https://www.computerhope.com/jargon/s/stdin.htm). (A word is a non-zero-length sequence of characters delimited by [white space](https://www.computerhope.com/jargon/w/whitspac.htm).)

The options below may be used to select which counts are printed. Counts are always in the following order: **newline**, **word**, **character**, **byte**, **maximum line length**.

## wc syntax

wc [*OPTION*]... [*FILE*]...

wc [*OPTION*]... --files0-from=*F*

* comm

## Description

Compare sorted files **FILE1** and **FILE2** line-by-line.

With no options, **comm** produces three-column output. Column one contains lines unique to **FILE1**, column two contains lines unique to **FILE2**, and column three contains lines common to both files. Each of these columns can be suppressed individually with options.

## comm syntax

comm [*OPTION*]... *FILE1* *FILE2*

## Examples

Let's say you have two text files, **recipe.txt** and **shopping-list.txt**.

**recipe.txt** contains these lines:

All-Purpose Flour

Baking Soda

Bread

Brown Sugar

Chocolate Chips

Eggs

Milk

Salt

Vanilla Extract

White Sugar

And **shopping-list.txt** contains these lines:

All-Purpose Flour

Bread

Brown Sugar

Chicken Salad

Chocolate Chips

Eggs

Milk

Onions

Pickles

Potato Chips

Soda Pop

Tomatoes

White Sugar

As you can see, the two files are different, but many of the lines are the same. Not all of the recipe ingredients are on the shopping list, and not everything on the shopping list is part of the recipe.

If we run the **comm** command on the two files, it will read both files and give us three columns of output:

comm recipe.txt shopping-list.txt

* **cmp**

**cmp** - Compare two files, and if they differ, tells the first byte and line number where they differ.

EXAMPLES

Compare two files

$ cat sample.txt

This is a sample text file

$ cat sample1.txt

This is another sample file

$ cmp sample.txt sample1.txt

sample.txt sample1.txt differ: byte 10, line 1

* diff

**diff** analyzes two files and prints the lines that are different. Essentially, it outputs a set of instructions for *how to change one file to make it identical to the second file.*

It does not actually change the files; however, it can optionally generate a script (with the **-e** option) for the program [**ed**](https://www.computerhope.com/unix/ued.htm) (or [**ex**](https://www.computerhope.com/unix/uex.htm) which can be used to apply the changes.

diff file1.txt file2.txt

Output:

2a3

> Oh yeah, I also need to buy grated cheese.

**Compressing and achieving files**

**Compressing files**

In \*nix land, bz and gz are two of the most common compression formats. Typically you use the bzip2 utility to create bz files and gzip to create gz. The fundamental difference is in the compression algorithm used by bzip2, which results in considerably smaller files. The downside is that bzip2 eats up more memory.

To compress a file using gzip, use the command gzip *filename*. The result is a file named *filename*.gz. Thus the command gzip homepage.htm yields homepage.htm.gz.

One thing to remember about gzip is that it replaces the original file with one which has .gz extension.

To uncompress files, use either gzip -d or gunzip.

bzip2 is similar to gzip. As with gzip, bzip2 also overwrites the original file with one which has a .bz or .bz2 extension. Decompressing .bz files is a breeze -- use bzip2 -d or bunzip2.

Both gzip and bzip2 maintain the ownership and permissions of the original file when compressing.

You can also use the zip utility to compress files, if you wish to share files with friends who use a non-\*nix platform. zip files.zip file1 file2 file3 would compress the three files, display the rate of compression of each file, and store them in files.zip. The unzip program can be used to extract the contents of a zip file.

**Compressed archives**

Unlike zip, which offers compression and archiving functionality, tar is capable of archiving only. This means that after you create a tarball, its size is the same as the cumulative size of the individual files. To reduce the size of a tarball, you must compress it by using either gzip or bzip2:

tar -cf archived.tar file1 file2 file3  
gzip archived.tar

This compresses archived.tar and replaces it with archived.tar.gz. You could also use bzip2 instead of gzip.

How do you extracting files from a compressed tarball? Use tar zxvf archived.tar.gz to extract all the files from a gzip-compressed tarball. The z switch tells tar that the tarball was compressed using gzip.

If you used bzip2 to compress this tarball, you'd get an error message if you used tar's z switch. To decompress a bzip2-compressed tarball, you need to use the j switch in its place. tar jxvf archived.tar.bz would extract the files.

**Assignment 4**

# **ls -l command in Linux**

ls -l option flag lists with long listing format.

### Syntax

$ ls -l [*options*] [*file*|*dir*]

### Examples

*ls*, default list with short format:

$ ls  
Desktop   Downloads Pictures Templates Videos  
Documents Music     Public   todo.txt  
$

File permissions

File ownership is an important component of Unix that provides a secure method for storing files. Every file in Unix has the following attributes −

* **Owner permissions** − The owner's permissions determine what actions the owner of the file can perform on the file.
* **Group permissions** − The group's permissions determine what actions a user, who is a member of the group that a file belongs to, can perform on the file.
* **Other (world) permissions** − The permissions for others indicate what action all other users can perform on the file.

## The Permission Indicators

While using **ls -l** command, it displays various information related to file permission as follows −

$ls -l /home/amrood

-rwxr-xr-- 1 amrood users 1024 Nov 2 00:10 myfile

drwxr-xr--- 1 amrood users 1024 Nov 2 00:10 mydir

Here, the first column represents different access modes, i.e., the permission associated with a file or a directory.

## Changing Ownership

The **chown** command changes the ownership of a file. The basic syntax is as follows −

$ chown user filelist

The value of the user can be either the **name of a user** on the system or the **user id (uid)** of a user on the system.

The following example will help you understand the concept −

$ chown amrood testfile

$

Changes the owner of the given file to the user **amrood**.

**NOTE** − The super user, root, has the unrestricted capability to change the ownership of any file but normal users can change the ownership of only those files that they own.

## Changing Group Ownership

The **chgrp** command changes the group ownership of a file. The basic syntax is as follows −

$ chgrp group filelist

The value of group can be the **name of a group** on the system or **the group ID (GID)** of a group on the system.

Following example helps you understand the concept −

$ chgrp special testfile

$

Changes the group of the given file to **special** group.

The permissions are broken into groups of threes, and each position in the group denotes a specific permission, in this order: read (r), write (w), execute (x) −

* The first three characters (2-4) represent the permissions for the file's owner. For example, **-rwxr-xr--** represents that the owner has read (r), write (w) and execute (x) permission.
* The second group of three characters (5-7) consists of the permissions for the group to which the file belongs. For example, **-rwxr-xr--**represents that the group has read (r) and execute (x) permission, but no write permission.
* The last group of three characters (8-10) represents the permissions for everyone else. For example, **-rwxr-xr--** represents that there is **read (r)** only permission.

## File Access Modes

The permissions of a file are the first line of defense in the security of a Unix system. The basic building blocks of Unix permissions are the **read**, **write**, and **execute** permissions, which have been described below −

### Read

Grants the capability to read, i.e., view the contents of the file.

### Write

Grants the capability to modify, or remove the content of the file.

### Execute

User with execute permissions can run a file as a program.

## Directory Access Modes

Directory access modes are listed and organized in the same manner as any other file. There are a few differences that need to be mentioned −

### Read

Access to a directory means that the user can read the contents. The user can look at the **filenames** inside the directory.

### Write

Access means that the user can add or delete files from the directory.

### Execute

Executing a directory doesn't really make sense, so think of this as a traverse permission.

A user must have **execute** access to the **bin** directory in order to execute the **ls** or the **cd** command.

Directory Permissions

hen applying permissions to directories on Linux, the permission bits have different meanings than on regular files.

* The write bit allows the affected user to create, rename, or delete files within the directory, and modify the directory's attributes
* The read bit allows the affected user to list the files within the directory
* The execute bit allows the affected user to enter the directory, and access files and directories inside
* The sticky bit states that files and directories within that directory may only be deleted or renamed by their owner (or root)

## Changing Permissions

To change the file or the directory permissions, you use the **chmod** (change mode) command. There are two ways to use chmod — the symbolic mode and the absolute mode.

### Using chmod in Symbolic Mode

The easiest way for a beginner to modify file or directory permissions is to use the symbolic mode. With symbolic permissions you can add, delete, or specify the permission set you want by using the operators in the following table.

|  |  |
| --- | --- |
| **S.No.** | **Chmod operator & Description** |
| 1 | **+**  Adds the designated permission(s) to a file or directory. |
| 2 | **-**  Removes the designated permission(s) from a file or directory. |
| 3 | **=**  Sets the designated permission(s). |

Absolute or Relative permissions

. **chmod command** can be use to change different permission configurations. chmod takes two lists as its arguments: permission changes and filenames.

You can specify the list of permissions in two different ways. One way uses permission symbols and is referred to as the symbolic method. The other uses what is known as a “binary mask” and is referred to as either the absolute or the relative method

## Using chmod with Absolute Permissions

The second way to modify permissions with the chmod command is to use a number to specify each set of permissions for the file.

Each permission is assigned a value, as the following table shows, and the total of each set of permissions provides a number for that set.

|  |  |  |
| --- | --- | --- |
| **Number** | **Octal Permission Representation** | **Ref** |
| **0** | No permission | --- |
| **1** | Execute permission | --x |
| **2** | Write permission | -w- |
| **3** | Execute and write permission: 1 (execute) + 2 (write) = 3 | -wx |
| **4** | Read permission | r-- |
| **5** | Read and execute permission: 4 (read) + 1 (execute) = 5 | r-x |
| **6** | Read and write permission: 4 (read) + 2 (write) = 6 | rw- |
| **7** | All permissions: 4 (read) + 2 (write) + 1 (execute) = 7 | rwx |

Here's an example using the testfile. Running **ls -1** on the testfile shows that the file's permissions are as follows −

$ls -l testfile

-rwxrwxr-- 1 amrood users 1024 Nov 2 00:10 testfile

**Assignment 5**

VI EDITOR AND ITS BASICS:

vi is generally considered the de facto standard in Unix editors because −

It&#39;s usually available on all the flavors of Unix system.

Its implementations are very similar across the board.

It requires very few resources.

It is more user-friendly than other editors such as the ed or the ex.

To Start vi

1. vi filename Create or Edit filename starting at line 1

2. vi -r filename Recover filename that was being edited

when system crashed

To Exit vi

1. :x&lt;return&gt; Quit vi, writing out modified file to file

named in original invocation

2. :wq&lt;return&gt; Quit vi, writing out modified file to file

named in original invocation

3. :q&lt;return&gt; Quit or exit vi

4. :q!&lt;return&gt; Quit vi even though latest changes have not

been saved for this vi call

Moving the cursor

1. j or &lt;return&gt; Move cursor down one line

[or down-arrow]

2. k[or up-arrow] Move cursor up one line

3. h or&lt;backspace&gt; Move cursor left one line

[or left-arrow]

l or&lt;space&gt;[or right-arrow] Move cursor right one line

4. 0(zero) Move cursor to start of current line(the one

with the arrow)

5. $ Move cursor to end of the current line

6. w Move cursor to beginning of next word

7. b Move cursor back to beginning of preceding

word

8. :0&lt;return&gt; or 1G Move cursor to first line in file

9. :n&lt;return&gt;or nG Move cursor to line n in file

10. :$&lt;return&gt;or G Move cursor to last line in file

Screen Manipulation

1. ^f Move forward one screen

2. ^b Move backward one screen

3. ^d Move down(forward) one half screen

4. ^u Move up(back)one half sreen

5. ^l Redraws the screen

6. ^r Redraws the screen,removing the deleted

lines

Adding,Changing,Deleting the text

1. u Undo whatever you just did(a simple toggle)

Inserting or Adding text

1. i Insert text before cursor,until&lt;esc&gt;hit

2. I Insert text at beginning of current

line,until&lt;esc&gt;hit

3. a Append text after cursor,until&lt;esc&gt;hit

4. A Append text to end of current

line,until&lt;esc&gt;hit

5. o Open and put text in a new line below

current line,until&lt;esc&gt;hit

6. O Open and put text in a new line above

current line,until&lt;esc&gt;hit

You can use the vi editor to edit an existing file or to create a new file from scratch.

You can also use this editor to just read a text file.

Operation Modes

While working with the vi editor, we usually come across the following two modes −

Command mode − This mode enables you to perform administrative tasks

such as saving the files, executing the commands, moving the cursor, cutting

(yanking) and pasting the lines or words, as well as finding and replacing. In

this mode, whatever you type is interpreted as a command.

Insert mode − This mode enables you to insert text into the file. Everything

that&#39;s typed in this mode is interpreted as input and placed in the file.

vi always starts in the command mode. To enter text, you must be in the insert mode

for which simply type i. To come out of the insert mode, press the Esc key, which will

take you back to the command mode.

REPEAT FACTOR

Many vi mode commands may be prefixed by a command repeat factor. This means

that most character, change, delete, word, movement, and positioning commands may

be preceded by a number which refers to the number of times the command should be

repeated. For example, j moves the cursor 1 character down, and 4j moves the cursor

4 characters down.

INPUT MODE AND INSERTION OF TEXT

Note that the repeat factor only works text insert mode, is also referred to as simply

&quot;insert mode&quot; or &quot;input mode.&quot; It is used for entering text into the buffer memory (and

simultaneously onto the screen). In this mode everything that is typed on the keyboard

is added to the text and does not become a command (although you can perform some

command operations in text mode with vi clones).with vi mode commands; it does

not work with ex commands

The most common way to switch from command mode to the input mode is to

use the i (which stands for &quot;insert&quot; or &quot;input&quot;) command.

SAVING TEXT AND QUITTING

The ex mode is an extension of command mode. To get into it, press Esc and

then : (the colon). The cursor will go to the bottom of the screen at a colon prompt.

Write your file by entering :w and quit by entering :q. You can combine these to save

and exit by entering :wq. However, if you&#39;re finished with your file, it&#39;s generally more

convenient to type Shift-z- z from command mode.

NAVIGATION

Cursor Navigation in vi editor

By character – h- left, j- down, k- up, l- right.

Screen Navigation in vi editor

Move window one line down – CTRL-E

Move window one line up – CTRL-Y

Move window one page down – CTRL-F

Move window one page up – CTRL-B

Mark position as n – mn

Go to marked position n – ‘n

EDITING TEXT

TABLE 4.1 vi Commands to Add Text

Command Function

a Adds text after the cursor

A Adds text at the end of the current line

i Inserts text before the cursor

I Inserts text at the beginning of the current line

Command Function

o Inserts a blank line after the current line

O Inserts a blank line before the current line

TABLE 4.2 vi Commands to Delete Text

Comm Function

and

x Deletes one character (under the cursor)

X Deletes one character (behind the cursor)

Deletes the current line

dd

Deletes five lines starting with the current line (any number would

5dd

work here)

Deletes the current word

dw

Changes the current word (deletes it and enters input mode)

cw

r Replaces the character under the cursor with the next character

you type

R Replaces the existing text with the text you type (like

overtype mode in most word processors)

TABLE 4.3 Other Handy vi Editing Commands

Command Function

yy Copies the current line

P Pastes any copied text after the cursor or line

J Joins the current and following lines

u Undoes the last change

U Undoes all changes on the current line

. Repeats the last command

UNDO LAST EDITING

vim undo (how to undo a change in vi/vim)

You “undo” changes in vi and vim with the undo command, which is the u key when

you are in vim command mode.

REPEATING LAST COMMAND

To repeat a command-line command, try @:, To repeat a normal/insert-

mode command, try .

SEARCHING FOR A PATTERN

Finding a Character String

A character string is one or more characters in succession. A string might include

letters, numbers, punctuation, special characters, blank spaces, tabs, or carriage returns.

A string can be a grammatical word or it can be part of a word.

To find a character string, type / followed by the string you want to search for,

and then press Return. vi positions the cursor at the next occurrence of the string.

For example, to find the string “meta,” type /meta followed by Return.

Type n to go to the next occurrence of the string. Type N to go

to the previous occurrence.

Certain special characters ( / &amp; ! . ^ \* $ \ ?) have special significance to the search

process and must be “escaped” when they are used in a search. To escape a special

character, precede it with a backslash (\).

Refining the Search

You can make searches more precise by tagging the string with indicators for

the following characteristics:

Beginning of

line End of line

Beginning of

word End of word

Wildcard characters

To match the beginning of a line, start the search string with a caret (^). For example,

to find the next line beginning with “Search”, type: /^Search

To match the end of a line, end the search string with a dollar sign ($). For example, to

find the next line ending with “search.”, type: /search\.$

Note that the period is escaped with a backslash.

To match the beginning of a word, type \&lt; at the beginning of the string; to match the

end of a word, type \&gt; at the end of the string. Thus, to match a word, rather than a

string, combine the end-of- word and beginning-of- word tags in the search pattern. For

example, to find the next occurrence of the word—as opposed to the string—“search”,

type: /\&lt;search\&gt;

To match any character, type a period (.) in the string at the location to be matched. For

example, to find the next occurrence of “disinformation” or “misinformation,” type:

/.isinformation

Because this is a search for a string and not a word, this search pattern might also find

such constructions as “misinformationalist” and “disinformationism.”

To match zero or more occurrences of the last character, type an asterisk (\*) in

the string. /[a-z]\*isinformation

SUBSTITUTION – SEARCH AND REPLACE

VI search and replace command format

Simple format is as follows:

%s/old-string/new- string/

VI search and replace command examples

Let us say you would like to find a word called “foo” and replace with

“bar”. First hit [Esc] key

Type : :%s/foo/bar/

Above command will replace first occurrence of word foo with bar on all lines. The %

is shorthand for all lines. To replace all occurrences of word foo with bar on all lines,

use the g option (which indicates all occurrences on a line). :%s/foo/bar/g

Note that the g can be replaced with a number 1,2,…N to change only the

n’th occurrence on each line.

Use find and replace on line ranges (match by line numbers)

You can also make changes on range of lines i.e. replace first occurrence of foo with

bar on lines 5 through 20 only, enter:

:5,20s/foo/bar/

Match by words:

Finally, you can match

**Assignment 6**

SJF

#include <iostream>  
using namespace std;  
int main()  
{  
  
int n,wt;  
  cout<<"Enter the number of processes";  
  cin>>n;  
  
int bursttime[30];  
    
for(int i=0;i<n;i++)  
{  
    
  cout<<"Enter the cpu burst time for process">>i+1;  
  cin>>arr1[i];  
  
}  
}  
  
void sjf()  
{  
  
for(i=1;i<n;++i)  
    {  
        for(j=0;j<(n-i);++j)  
            if(arr1[j]>arr1[j+1])  
            {  
                temp=arr1[j];  
                arr1[j]=arr1[j+1];  
                arr1[j+1]=temp;  
            }  
    }  
}

FCFS

#include<iostream>

using namespace std;

int main()

{

    int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;

    cout<<"Enter total number of processes(maximum 20):";

    cin>>n;

    cout<<"\nEnter Process Burst Time\n";

    for(i=0;i<n;i++)

    {

        cout<<"P["<<i+1<<"]:";

        cin>>bt[i];

    }

    wt[0]=0;    //waiting time for first process is 0

    //calculating waiting time

    for(i=1;i<n;i++)

    {

        wt[i]=0;

        for(j=0;j<i;j++)

            wt[i]+=bt[j];

    }

    cout<<"\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time";

    //calculating turnaround time

    for(i=0;i<n;i++)

    {

        tat[i]=bt[i]+wt[i];

        avwt+=wt[i];

        avtat+=tat[i];

        cout<<"\nP["<<i+1<<"]"<<"\t\t"<<bt[i]<<"\t\t"<<wt[i]<<"\t\t"<<tat[i];

    }

    avwt/=i;

    avtat/=i;

    cout<<"\n\nAverage Waiting Time:"<<avwt;

    cout<<"\nAverage Turnaround Time:"<<avtat;

    return 0;

}

Round Robin Scheduling

// C++ program for implementation of RR scheduling

#include<iostream>

using namespace std;

// Function to find the waiting time for all

// processes

void findWaitingTime(int processes[], int n,

             int bt[], int wt[], int quantum)

{

    // Make a copy of burst times bt[] to store remaining

    // burst times.

    int rem\_bt[n];

    for (int i = 0 ; i < n ; i++)

        rem\_bt[i] =  bt[i];

    int t = 0; // Current time

    // Keep traversing processes in round robin manner

    // until all of them are not done.

    while (1)

    {

        bool done = true;

        // Traverse all processes one by one repeatedly

        for (int i = 0 ; i < n; i++)

        {

            // If burst time of a process is greater than 0

            // then only need to process further

            if (rem\_bt[i] > 0)

            {

                done = false; // There is a pending process

                if (rem\_bt[i] > quantum)

                {

                    // Increase the value of t i.e. shows

                    // how much time a process has been processed

                    t += quantum;

                    // Decrease the burst\_time of current process

                    // by quantum

                    rem\_bt[i] -= quantum;

                }

                // If burst time is smaller than or equal to

                // quantum. Last cycle for this process

                else

                {

                    // Increase the value of t i.e. shows

                    // how much time a process has been processed

                    t = t + rem\_bt[i];

                    // Waiting time is current time minus time

                    // used by this process

                    wt[i] = t - bt[i];

                    // As the process gets fully executed

                    // make its remaining burst time = 0

                    rem\_bt[i] = 0;

                }

            }

        }

        // If all processes are done

        if (done == true)

          break;

    }

}

// Function to calculate turn around time

void findTurnAroundTime(int processes[], int n,

                        int bt[], int wt[], int tat[])

{

    // calculating turnaround time by adding

    // bt[i] + wt[i]

    for (int i = 0; i < n ; i++)

        tat[i] = bt[i] + wt[i];

}

// Function to calculate average time

void findavgTime(int processes[], int n, int bt[],

                                     int quantum)

{

    int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

    // Function to find waiting time of all processes

    findWaitingTime(processes, n, bt, wt, quantum);

    // Function to find turn around time for all processes

    findTurnAroundTime(processes, n, bt, wt, tat);

    // Display processes along with all details

    cout << "Processes "<< " Burst time "

         << " Waiting time " << " Turn around time\n";

    // Calculate total waiting time and total turn

    // around time

    for (int i=0; i<n; i++)

    {

        total\_wt = total\_wt + wt[i];

        total\_tat = total\_tat + tat[i];

        cout << " " << i+1 << "\t\t" << bt[i] <<"\t "

             << wt[i] <<"\t\t " << tat[i] <<endl;

    }

    cout << "Average waiting time = "

         << (float)total\_wt / (float)n;

    cout << "\nAverage turn around time = "

         << (float)total\_tat / (float)n;

}

// Driver code

int main()

{

    // process id's

    int processes[] = { 1, 2, 3};

    int n = sizeof processes / sizeof processes[0];

    // Burst time of all processes

    int burst\_time[] = {10, 5, 8};

    // Time quantum

    int quantum = 2;

    findavgTime(processes, n, burst\_time, quantum);

    return 0;

}

**Assignment 7**

1. **r/w/x permissions for directory:-**

r permission:- If your directory has only r (read) permission you are allowed to see the content of the directory. But we cannot write or execute in the read permission in the directory.

w permission:-This gives user the permission to write in a directory. When the user is given the write permission on CLI, a text box will appear where the data can be written which will be stored in the file.

x permission:-This command means execution. When the x command is run on CLI in linux, the user can access any file present in the directory.

**2.Touch command:-** The *touch* command is the easiest way to create new, empty files. It is also used to change the *timestamps* (i.e., dates and times of the most recent access and modification) on existing files and directories.

Touch's syntax is:-touch [option] file name(s)

When used without any options, touch creates new files for any file names that are provided as *arguments* (i.e., input data) if files with such names do not already exist. Touch can create any number of files simultaneously.

Thus, for example, the following command would create three new, empty files named *file1*, *file2* and *file3*:

touch file1 file2 file3

3.Find command:- In Unix-like and some other operating systems, find is a command-line utility that searches one or more directory trees of a file system, locates files based on some user-specified criteria and applies a user-specified action on each matched file. Its syntax is:-

Find options starting/path expression

* The options attribute will control the behavior and optimization method of the find process.
* The starting/path attribute will define the top level directory where find begins filtering.
* The expression attribute controls the tests that search the directory hierarchy to produce output.

**4.Unmask command:-** In computing, umask is a command that determines the settings of a mask that controls how file permissions are set for newly created files. It also may refer to a function that sets the mask, or it may refer to the mask itself, which is formally known as the file mode creation mask. The mask is a grouping of bits, each of which restricts how its corresponding permission is set for newly created files. The bits in the mask may be changed by invoking the umask command.Its syntax is:-

$ unmask –S:-display current value symbolically

**5.Hard link:-** A *hard link* is merely an additional name for an existing file on Linux or other Unix-like operating systems. Any number of hard links, and thus any number of names, can be created for any file. Hard links can also be created to other hard links. However, they cannot be created for directories, and they cannot cross file system boundaries or span across partitions. Hard links are created with the *ln* command. For example, the following would create a hard link named *hlink1* to a file named *file1*, both in the current directory (i.e., the directory in which the user is currently working):

ln file1 hlink1

**6.Symbolic link:-** To make links between files you need to use ln command. A symbolic link (also known as a soft link or symlink) consists of a special type of file that serves as a reference to another file or directory. Unix/Linux like operating systems often uses symbolic links. It is also known as soft link. Soft links are created with the ln command. For example, the following would create a soft link named link1 to a file named file1, both in the current directory

$ ln –s file1 link1

To verify new soft link run:

$ ln -l file1 link1.

**7.Permission:-** File systems use permissions and attributes to regulate the level of interaction that system processes can have with files and directories. Chmod is a command in Linux and other Unix-like operating systems that allows to *ch*ange the permissions (or access *mod*e) of a file or directory. In unix type operating system, there are read, write and execute permissions in files and directories which are mainly used. More types of permissions are listed below:-

(a)u:- the user owns a file.

(b)g:- the group that the file belongs to.

(c)o:-the other users means everyone.

(d)a:- all of the above; use this instead of typing ugo.

**8.Modification and access timings:-**You can change the modification time of a file using the  command: touch file name

If you want to modify the file relative to its existing modification time instead, the following should do the trick:

touch -d "$(date -R -r filename) - 2 hours" filename

You can change the arguments to find to select only the files you are interested in. If you only want to update the file modification times relative to the present time, you can simplify this to:

find DIRECTORY -exec touch -d "2 hours ago" {}

**9.Locating files:-**Find is a command for recursively filtering objects in the file system based on a simple conditional mechanism. Use find to search for a file or directory on your file system. Using the -exec flag, files can be found and immediately processed within the same command. The find command is only able to filter the directory hierarchy based on a file’s name and meta data. If you need to search based on the content of the file, use a tool like grep. Consider the following example:

Find . – type f –exec grep “example” ‘[]’ \;

**10.Expr:-** Expr is a command line Unix utility which evaluates an expression and outputs the corresponding value. Expr evaluates integer or string expressions, including pattern matching regular expressions. The operators available for integers: addition, subtraction, multiplication, division and modulus for strings: find regular expression, find a set of characters in a string; in some versions: find substring, length of string for either: comparison (equal, not equal, less than, etc.). For example:-

To perform addition of two numbers:

$ expr 3 + 5

output:8

|  |
| --- |
|  |