sourcemind

Linux Basics

Mocktar ISSA
Full Stack Software Engineer

sourcemind

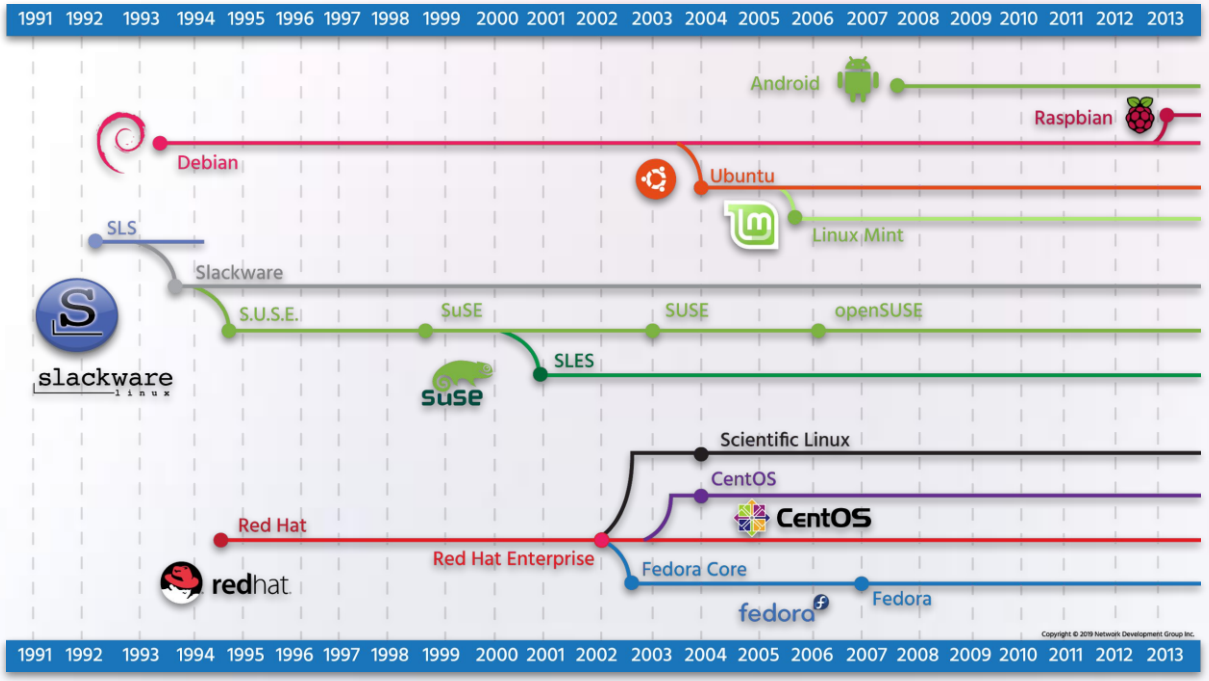
Linux Distributions

A Linux distribution is a bundle of software, typically comprised of the Linux kernel, utilities, management tools, and even some application software in a package which also includes the means to update core software and install additional applications. The number of Linux distributions available numbers in the hundreds, so the choice can seem daunting at first. However, the decision points are mostly the same as those highlighted for choosing an operating system.

- Role
- Function
- Life Cycle
- Stability
- Cost
- Interface
  - graphical (GUI)
  - non-graphical (CLI)

Most usable Linux Distributions

- Red Hat
- Debian
- SUSE
- Android
- Raspbian



https://upload.wikimedia.org/wikipedia/commons/1/1b/Linux_Distribution_Timeline.svg

sourcemind

Virtualization

Virtualization is technology that allows you to create multiple simulated environments or dedicated resources from a single, physical hardware system. Software called a hypervisor connects directly to that hardware and allows you to split 1 system into separate, distinct, and secure environments known as *virtual machines (VMs).* These VMs rely on the *hypervisor's* ability to separate the machine's resources from the hardware and distribute them appropriately. Virtualization helps you get the most value from previous investments.

The physical hardware, equipped with a *hypervisor*, is called the *host*, while the many VMs that use its resources are *guests*. These guests treat computing resources like CPU, memory, and storage as a pool of resources that can easily be relocated. Operators can control virtual instances of CPU, memory, storage, and other resources, so guests receive the resources they need when they need them.

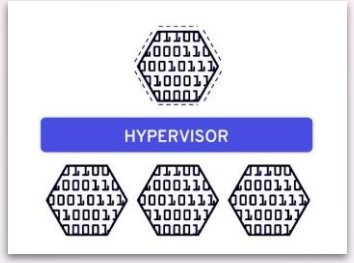
Resources are partitioned as needed from the physical environment to the many virtual environments. Users interact with and run computations within the virtual environment (typically called a guest machine or virtual machine). The virtual machine functions as a single data file. And like any digital file, it can be moved from one computer to another, opened in either one, and be expected to work the same.

When the virtual environment is running and a user or program issues an instruction that requires additional resources from the physical environment, the hypervisor relays the request to the physical system and caches the changes which all happens at close to native speed (particularly if the request is sent through an open source hypervisor based on KVM, the Kernel-based Virtual Machine).

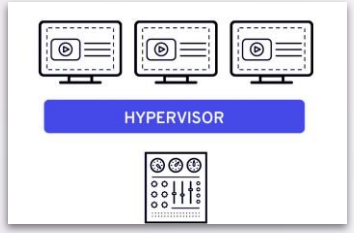
sourcemind

Types of Virtualization

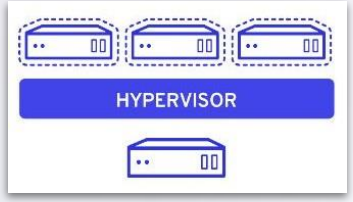
Data Virtualization: Data that's spread all over can be consolidated into a single source. Data virtualization allows companies to treat data as a dynamic supply—providing processing capabilities that can bring together data from multiple sources, easily accommodate new data sources, and transform data according to user needs. Data virtualization tools sit in front of multiple data sources and allows them to be treated as single source, delivering the needed data—in the required form—at the right time to any application or user.



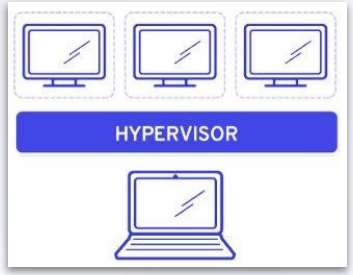
Desktop Virtualization: Easily confused with operating system virtualization—which allows you to deploy multiple operating systems on a single machine—desktop virtualization allows a central administrator (or automated administration tool) to deploy simulated desktop environments to hundreds of physical machines at once. Unlike traditional desktop environments that are physically installed, configured, and updated on each machine, desktop virtualization allows admins to perform mass configurations, updates, and security checks on all virtual desktops.



Server Virtualization: Servers are computers designed to process a high volume of specific tasks really well so other computers—like laptops and desktops—can do a variety of other tasks. Virtualizing a server lets it to do more of those specific functions and involves partitioning it so that the components can be used to serve multiple functions.



Operating system virtualization: Operating system virtualization happens at the kernel the central task managers of operating systems. It's a useful way to run Linux and Windows environments side-by-side. Enterprises can also push virtual operating systems to computers, which:



- Reduces bulk hardware costs, since the computers don't require such high out-of-the-box capabilities.
- Increases security, since all virtual instances can be monitored and isolated.
- Limits time spent on IT services like software updates.

sourcemind

Linux Kernel

The Linux kernel is the main component of a Linux operating system (OS) and is the core interface between a computer's hardware and its processes. It communicates between the 2, managing resources as efficiently as possible.
The kernel is so named because-like a seed inside a hard shell-it exists within the OS and controls all the major functions of the hardware, whether it's a phone, laptop, server, or any other kind of computer.

What the kernel does

The kernel has 4 jobs:

1. *Memory management:* Keep track of how much memory is used to store what, and where
2. *Process management:* Determine which processes can use the central processing unit (CPU), when, and for how long
3. *Device drivers:* Act as mediator/interpreter between the hardware and processes
4. *System calls and security:* Receive requests for service from the processes

The kernel, if implemented properly, is invisible to the user, working in its own little world known as kernel space, where it allocates memory and keeps track of where everything is stored. What the user sees-like web browsers and files-are known as the user space. These applications interact with the kernel through a system call interface (SCI).

Think about it like this. The kernel is a busy personal assistant for a powerful executive (the hardware). It's the assistant's job to relay messages and requests (processes) from employees and the public (users) to the executive, to remember what is stored where (memory), and to determine who has access to the executive at any given time and for how long.

https://www.redhat.com/en/topics/linux/what-is-the-linux-kernel

sourcemind

Applications and Package Management

Applications: Applications make requests to the kernel and in return receive resources, such as memory, CPU, and disk space. If two applications request the same resource, the kernel decides which one gets it, and in some cases, kills off another application to save the rest of the system and prevent a crash.
Linux software generally falls into one of three categories:

- *Server Applications:* Software that has no direct interaction with the monitor and keyboard of the machine it runs on. Its purpose is to serve information to other computers, called clients. Sometimes server applications may not talk to other computers but only sit there and crunch data. [Apache, MySQL, Email Servers]
- *Desktop Applications:* Web browsers, text editors, music players, or other applications with which users interact directly. In many cases, such as a web browser, the application is talking to a server on the other end and interpreting the data. This is the “client” side of a client/server application. [LibreOffice, Web Browsers]
- *Tools:* A category of software that exists to make it easier to manage computer systems. Tools can help configure displays, provide a Linux shell that users type commands into, or even more sophisticated tools, called compilers, that convert source code to application programs that the computer can execute. [Shells, text editors]

Package Management: Every Linux system needs to add, remove, and update software. In the past this meant downloading the source code, setting it up, compiling it, and copying files onto each system that required updating. Thankfully, modern distributions use packages, which are compressed files that bundle up an application and its dependencies (or required files), greatly simplifying the installation by making the right directories, copying the proper files into them, and creating such needed items as symbolic links.

Debian Package Management: The Debian distribution, and its derivatives such as Ubuntu and Mint, use the Debian package management system. At the heart of Debian package management are software packages that are distributed as files ending in the .deb extension. The lowest-level tool for managing these files is the command. This command can be tricky for novice Linux users, so the *Advanced Package Tool*, *apt-get* (a front-end program to the *dpkg* tool), makes management of packages easier.

RPM Package Management: The Linux Standards Base, which is a Linux Foundation project, is designed to specify (through a consensus) a set of standards that increase the compatibility between conforming Linux systems. According to the Linux Standards Base, the standard package management system is RPM. RPM makes use of an .rpm file for each software package.This system is what distributions derived from Red Hat, including Centos and Fedora, use to manage software. Several other distributions that are not Red Hat derived, such as SUSE, OpenSUSE, and Arch, also use RPM.

sourcemind

Bash Shell

CLI: The Linux community is different in that it positively celebrates the CLI for its power, speed, and ability to accomplish a vast array of tasks with a single command-line instruction. When a user first encounters the CLI, they can find it challenging because it requires memorizing a dizzying amount of commands and their options.

Advantage: Once a user has learned the structure of how commands are used, where the necessary files and directories are located, and how to navigate the hierarchy of a file system, they can be immensely productive. This capability provides more precise control, greater speed, and the ability to automate tasks more easily through scripting. Furthermore, by learning the CLI, a user can easily be productive almost instantly on ANY flavor or distribution of Linux, reducing the amount of time needed to familiarize themselves with a system because of variations in a GUI.

Shell: Once a user has entered a command the terminal then accepts what the user has typed and passes it to a shell. The shell is the command line interpreter that translates commands entered by a user into actions to be performed by the operating system. If output is produced by the command, then text is displayed in the terminal. If problems with the command are encountered, an error message is displayed.

Advantage:

- *Scripting: The ability to place commands in a file and then interpret (effectively use Bash to execute the contents of) the file, resulting in all of the commands being executed. This feature also has some programming features, such as conditional statements and the ability to create functions (AKA subroutines).*
- *Aliases: The ability to create short nicknames for longer commands.*
- *Variables: Used to store information for the Bash shell and for the user. These variables can be used to modify how commands and features work as well as provide vital system information.*

sourcemind

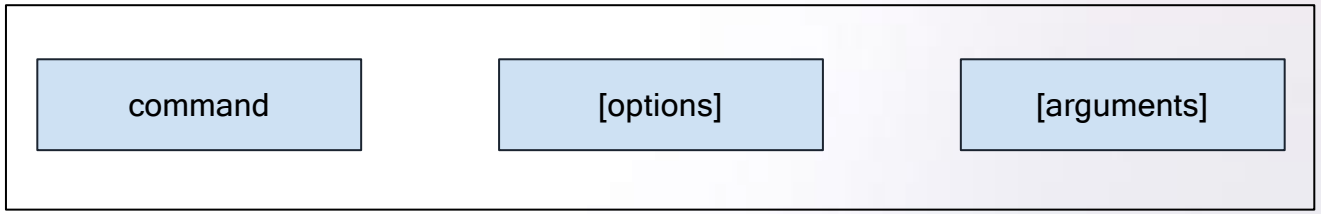
Commands

What is a command? The simplest answer is that a command is a software program that, when executed on the CLI, performs an action on the computer.

Many commands can be used by themselves with no further input. Some commands require additional input to run correctly. This additional input comes in two forms: *options* and *arguments.*

options: *Options are used to modify the core behavior of a command*

arguments: *Arguments are used to provide additional information (such as a filename or a username)*



Example:

```
adavtyan@artur-lpt:~$ ls -l Hello
-rw-rw-r-- 1 adavtyan adavtyan 0 Սեպ 16 18:28 Hello
adavtyan@artur-lpt:~$
```

Note: Keep in mind that Linux is case-sensitive. Commands, options, arguments, variables, and file names must be entered exactly as shown.

sourcemind

Commands

Internal commands: Also called built-in commands, internal commands are built into the shell itself. A good example is the *cd* (change directory) command as it is part of the Bash shell. When a user types the cd command, the Bash shell is already executing and knows how to interpret it, requiring no additional programs to be started.

Example:

```
adavtyan@artur-lpt:~$ type cd
cd is a shell builtin
adavtyan@artur-lpt:~$
```

External commands: Also called built-in commands, internal commands are built into the shell itself. A good example is the cd (change directory) command as it is part of the Bash shell. When a user types the *cd* command, the Bash shell is already executing and knows how to interpret it, requiring no additional programs to be started. If a command does not behave as expected or if a command is not accessible that should be, it can be beneficial to know where the shell is finding the command or which version it is using. It would be tedious to have to manually look in each directory that is listed in the PATH variable. Instead, use the *which* command to display the full path to the command in question:

Example:

```
adavtyan@artur-lpt:~$ which ls
/bin/ls
adavtyan@artur-lpt:~$ which cal
/usr/bin/cal
adavtyan@artur-lpt:~$ /bin/ls Hello
Hello
adavtyan@artur-lpt:~$ /usr/bin/cal
    Մայիսի 2021
Կր Եր Եր Չր Հև Ոև Շբ
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

sourcemind

Variables

Variables: A variable is a feature that allows the user or the shell to store data. This data can be used to provide critical system information or to change the behavior of how the Bash shell (or other commands) work. Variables are given names and stored temporarily in memory. There are two types of variables used in the Bash shell: local and environment.

Local Variables: Local or shell variables exist only in the current shell, and cannot affect other commands or applications. When the user closes a terminal window or shell, all of the variables are lost.

To set the value of a variable, use the following assignment expression. If the variable already exists, the value of the variable is modified. If the variable name does not already exist, the shell creates a new local variable and sets the value:

```
variable = value
```

Example:

```
adavtyan@artur-lpt:~$ variable1=Hello
adavtyan@artur-lpt:~$ echo $variable1
Hello
adavtyan@artur-lpt:~$
```

sourcemind

Variables

Environment Variables: Environment variables, also called global variables, are available system-wide, in all shells used by Bash when interpreting commands and performing tasks. The system automatically recreates environment variables when a new shell is opened. Examples include the PATH, HOME, and HISTSIZE variables.

Example:

```
adavtyan@artur-lpt:~$ echo $HOME
/home/adavtyan
adavtyan@artur-lpt:~$
```

When run without arguments, the *env* command outputs a list of the environment variables.

```
adavtyan@artur-lpt:~$ env | grep "HOME"
JAVA_HOME=/home/adavtyan/.sdkman/candidates/java/current
HOME=/home/adavtyan
adavtyan@artur-lpt:~$
```

The *export* command is used to turn a local variable into an environment variable.

```
adavtyan@artur-lpt:~$ export variable1='True'
adavtyan@artur-lpt:~$ env | grep variable1
variable1=True
adavtyan@artur-lpt:~$
```

Exported variables can be removed using the *unset* command:

```
adavtyan@artur-lpt:~$ unset variable1
adavtyan@artur-lpt:~$
```

sourcemind

Aliases

Aliases: An alias can be used to map longer commands to shorter key sequences. When the shell sees an alias being executed, it substitutes the longer sequence before proceeding to interpret commands.

To determine what aliases are set on the current shell use the *alias* command:

Example:

```
adavtyan@artur-lpt:~$ alias
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo error)" "$(history|tail -n1|sed -e '\''s/^\s*[0-9]\+\s*//;s/[;&|]\s*alert$//'\'')"'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
alias mc='~/mc'
adavtyan@artur-lpt:~$
```

New aliases can be created using the following format, where name is the name to be given the alias and command is the command to be executed when the alias is run.

Alias name = command

Example:

```
adavtyan@artur-lpt:~$ alias mycal="cal 2021"
adavtyan@artur-lpt:~$ mycal
```

sourcemind

Quoting

Quoting: Quotation marks are used throughout Linux administration and most computer programming languages to let the system know that the information contained within the quotation marks should either be ignored or treated in a way that is very different than it would normally be treated. There are three types of quotes that have special significance to the Bash shell: double quotes ", single quotes ', and back quotes `. Each set of quotes alerts the shell not to treat the text within the quotes in the normal way.

Double Quoting: Double quotes stop the shell from interpreting some metacharacters (special characters), including glob characters. Within double quotes an asterisk is just an asterisk, a question mark is just a question mark, and so on, which is useful when you want to display something on the screen that is normally a special character to the shell. In the echo command below, the Bash shell doesn't convert the glob pattern into filenames that match the pattern:

Example:

```
adavtyan@artur-lpt:~$ echo "The glob characters are *, ? and []"
The glob characters are *, ? and []
adavtyan@artur-lpt:~$
adavtyan@artur-lpt:~$ echo "The path is $PATH"
The path is /home/adavtyan/.kubectx:/home/adavtyan/.sdkman/candidates/java/current/bin:/home/adavtyan/.kubectx:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/game
s:/snap/bin:/home/adavtyan/istio-1.7.0/bin:/home/adavtyan/istio-1.7.0/bin
adavtyan@artur-lpt:~$
```

Single Quoting: Single quotes prevent the shell from doing any interpreting of special characters, including globs, variables, command substitution and other metacharacters that have not been discussed yet. For example, to make the $ character simply mean a $, rather than it acting as an indicator to the shell to look for the value of a variable, execute the second command displayed below:

Example:

```
adavtyan@artur-lpt:~$ echo The car costs $10000
The car costs 0000
adavtyan@artur-lpt:~$ echo 'The car costs $10000'
The car costs $10000
adavtyan@artur-lpt:~$
```

sourcemind

Backslash Character

Backslash Character: There is also an alternative technique to essentially single quote a single character. Consider the following message:

If this sentence is placed in double quotes, $1 and $PATH are considered variables.

```
adavtyan@artur-lpt:~$ echo "The service costs $2 and the pat is $PATH"
The service costs  and the pat is /home/adavtyan/.kubectx:/home/adavtyan/.sdkman/candidates/java/current/bin:/home/adavtyan/.kubectx:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr
/games:/usr/local/games:/snap/bin:/home/adavtyan/istio-1.7.0/bin:/home/adavtyan/istio-1.7.0/bin
adavtyan@artur-lpt:~$
```

If it is placed in single quotes, $1 and $PATH are not considered variables.

```
adavtyan@artur-lpt:~$ echo 'The service costs $2 and the path is $PATH'
The service costs $2 and the path is $PATH
adavtyan@artur-lpt:~$
```

But what if you want to have $PATH treated as a variable and $1 not?

In this case, use a backslash \ character in front of the dollar sign $ character to prevent the shell from interpreting it. The command below demonstrates using the \ character:

```
adavtyan@artur-lpt:~$ echo The service costs \$2 and the path is $PATH
The service costs $2 and the path is /home/adavtyan/.kubectx:/home/adavtyan/.sdkman/candidates/java/current/bin:/home/adavtyan/.kubectx:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/
usr/games:/usr/local/games:/snap/bin:/home/adavtyan/istio-1.7.0/bin:/home/adavtyan/istio-1.7.0/bin
adavtyan@artur-lpt:~$
```

sourcemind

Man Pages

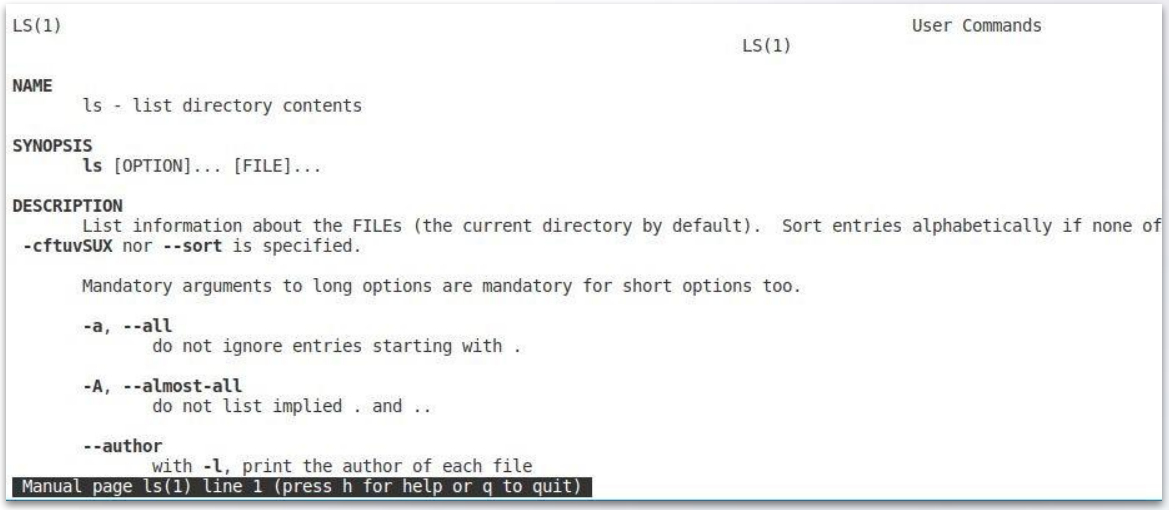
UNIX is the operating system that Linux was modeled after. The developers of UNIX created help documents called man pages (short for manual pages).

Man pages are used to describe the features of commands. They provide a basic description of the purpose of the command, as well as details regarding available options.

To view a man page for a command, use the man command:

```
man command
```

Navigate the document using the arrow keys:



sourcemind

Man Pages Categorized by Sections

Until now, we have been displaying man pages for commands. However, there are several different types of commands (user commands, system commands, and administration commands), configuration files and other features, such as libraries and kernel components, that require documentation.

By default, there are nine sections of man pages:

- General Commands
- System Calls
- Library Calls
- Special Files
- File Formats and Conventions
- Games
- Miscellaneous
- System Administration Commands
- Kernel Routines

```
adavtyan@artur-lpt:~$ man -f passwd
passwd (5)           - the password file
passwd (1ssl)        - compute password hashes
passwd (1)           - change user password
adavtyan@artur-lpt:~$ man passwd
adavtyan@artur-lpt:~$ man 5 passwd
adavtyan@artur-lpt:~$
```

Unfortunately, you won't always remember the exact name of the man page that you want to view. Fortunately, each man page has a short description associated with it. The -k option to the man command searches both the names and descriptions of the man pages for a keyword.

For example, to find a man page that displays how to copy directories, search for the keyword copy:

```
adavtyan@artur-lpt:~$ man -k copy
bcopy (3)            - copy byte sequence
COPY (7)             - copy data between a file and a table
copy_file_range (2)  - Copy a range of data from one file to another
```

sourcemind

Info Documentation

Man pages are excellent sources of information, but they do tend to have a few disadvantages. One example is that each man page is a separate document, not related to any other man page. While some man pages have a SEE ALSO section that may refer to other man pages, they tend to be independent sources of documentation.

The info command also provides documentation on operating system commands and features. The goal is slightly different from man pages: to provide a documentation resource that gives a logical organizational structure, making reading documentation easier.

All of the documentation is merged into a single "book" representing all of the documentation available. Within info documents, information is broken down into categories that work much like a table of contents in a book. Hyperlinks are provided to pages with information on individual topics for a specific command or feature.

Another advantage of info over man pages is that the writing style of info documents is typically more conducive to learning a topic. Consider man pages to be more of a reference resource and info documents to be more of a learning guide.

To display the info documentation for a command, use the info command.

```
info command
```

For example, to display the info page of the ls command:

```
adavtyan@artur-lpt:~$ info ls
```

Note: Info is the default format for documentation inside the GNU project, man is the much older traditional format for UNIX.

sourcemind

Using the help Option

Many commands will provide basic information, very similar to the SYNOPSIS found in man pages, by simply using the *--help* option to the command. This option is useful to learn the basic usage of a command quickly without leaving the command line:

```
adavtyan@artur-lpt:~$ cat --help
Usage: cat [OPTION]... [FILE]...
Concatenate FILE(s) to standard output.

With no FILE, or when FILE is -, read standard input.

  -A, --show-all           equivalent to -vET
  -b, --number-nonblank    number nonempty output lines, overrides -n
  -e                       equivalent to -vE
  -E, --show-ends          display $ at end of each line
  -n, --number             number all output lines
  -s, --squeeze-blank      suppress repeated empty output lines
  -t                       equivalent to -vT
  -T, --show-tabs          display TAB characters as ^I
  -u                       (ignored)
  -v, --show-nonprinting   use ^ and M- notation, except for LFD and TAB
      --help     display this help and exit
      --version  output version information and exit

Examples:
  cat f - g  Output f's contents, then standard input, then g's contents.
  cat        Copy standard input to standard output.

GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
Full documentation at: <https://www.gnu.org/software/coreutils/cat>
or available locally via: info '(coreutils) cat invocation'
adavtyan@artur-lpt:~$
```

sourcemind

Finding Commands and Documentation

The whatis command (or man -f) returns what section a man page is stored in. This command occasionally returns unusual output, such as the following:

```
adavtyan@artur-lpt:~$ whatis ls
ls (1)               - list directory contents
adavtyan@artur-lpt:~$
```

Based on this output, there are two ls commands that list directory contents. The simple reason for this is that UNIX had two main variants, which resulted in some commands being developed "in parallel" and therefore behaving differently on different variants of UNIX. Many modern distributions of Linux include commands from both UNIX variants.

This does, however, pose a bit of a problem: when the ls command is typed, which command is executed?

Where Are These Commands Located?

```
adavtyan@artur-lpt:~$ whereis ls
ls: /bin/ls /usr/share/man/man1/ls.1.gz
adavtyan@artur-lpt:~$
```

Man pages are easily distinguished from commands as they are typically compressed with a program called gzip, resulting in a filename that ends in .gz. Notice there are two man pages listed above, but only one command: /bin/ls. This is because the ls command can be used with the options/features that are described by either man page. So, when learning what can be done with the ls command, it may be interesting to explore both man pages. Fortunately, this is more of an exception as most commands only have one man page.

sourcemind

Find Any File or Directory

The `whereis` command is specifically designed to find commands and man pages. While this is useful, it is often necessary to find a file or directory, not just files that are commands or man pages.

To find any file or directory, use the `locate` command. This command searches a database of all files and directories that were on the system when the database was created. Typically, the command to generate this database is run nightly.

```
adavtyan@artur-lpt:~$ locate passwd
/etc/passwd
/etc/passwd-
/etc/cron.daily/passwd
/etc/pam.d/chpasswd
/etc/pam.d/passwd
```

In many cases, it is helpful to start by finding out how many files match. Do this by using the -c option to the locate command:

```
adavtyan@artur-lpt:~$ locate -c passwd
147
adavtyan@artur-lpt:~$
```

To limit the output produced by the locate command use the -b option. This option only includes listings that have the search term in the basename of the filename. The basename is the portion of the filename not including the directory names.

```
adavtyan@artur-lpt:~$ locate -c -b passwd
127
adavtyan@artur-lpt:~$
```

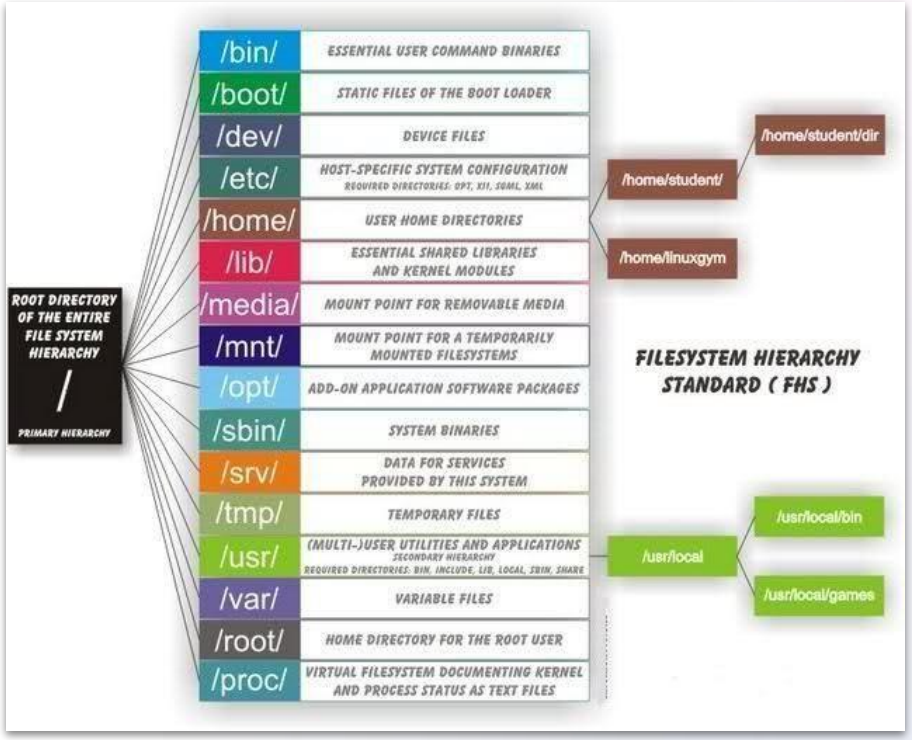
As you can see from the previous output, there will still be many results when the -b option is used. To limit the output even further, place a \ character in front of the search term. This character limits the output to filenames that exactly match the term

```
adavtyan@artur-lpt:~$ locate -b "\passwd"
/etc/passwd
/etc/cron.daily/passwd
```

sourcemind

Directory Structure

The following image shows a visual representation of a typical Linux filesystem:



To view the contents of the root directory, use the ls command with the / character as the argument:

```
adavtyan@artur-lpt:~$ ls /
bin  boot  cdrom  dev  etc  home  initrd.img  initrd.img.old  lib  lib64  lost+found  media  mnt  opt  proc  root  run  sbin  snap  srv  sys  tmp  usr  var  vmlinuz  vmlinuz.old
adavtyan@artur-lpt:~$
```

sourcemind

Paths

There are two types of paths: absolute and relative

Absolute Paths: Absolute paths allow the user to specify the exact location of a directory. It always starts at the root directory, and therefore it always begins with the / character. The path /home/sysadmin is an absolute path; it tells the system to begin at the root / directory, move into the home directory, and then into the sysadmin directory.

If the path /home/adavtyan is used as an argument to the cd command, it moves the user into the home directory for the adavtyan user.

```
adavtyan@artur-lpt:~/Downloads$ cd /home/adavtyan/
adavtyan@artur-lpt:~$ pwd
/home/adavtyan
adavtyan@artur-lpt:~$
```

Relative Paths: Relative paths start from the current directory. A relative path gives directions to a file relative to the current location in the filesystem. They do not start with the / character. Instead, they start with the name of a directory. More specifically, relative paths start with the name of a directory contained within the current directory.

```
adavtyan@artur-lpt:~$ cd Documents/
adavtyan@artur-lpt:~/Documents$ cd curse-WFA/
adavtyan@artur-lpt:~/Documents/curse-WFA$ cd Art/
adavtyan@artur-lpt:~/Documents/curse-WFA/Art$ cd ../../../D
Desktop/   Documents/ Downloads/
adavtyan@artur-lpt:~/Documents/curse-WFA/Art$ cd ../../../Downloads/
adavtyan@artur-lpt:~/Downloads$ pwd
/home/adavtyan/Downloads
adavtyan@artur-lpt:~/Downloads$
```

sourcemind

Listing Files and Directories

This ls command is used to display the contents of a directory and can provide detailed information about the files. By default, when it is used with no options or arguments, it lists the files in the current directory:

```
adavtyan@artur-lpt:~$ ls /var
backups  cache  crash  lib  local  lock  log  mail  metrics  opt  run  snap  spool  tmp  www
adavtyan@artur-lpt:~$
```

Listing Hidden Files:To display all files, including hidden files, use the -a option to the ls command:

```
adavtyan@artur-lpt:~$ sudo ls -a /root/
.  ..  .bash_history  .bashrc  .cache  .config  .dbus  .gnupg  .local  .openfortigui  .profile  .rpmdb  snap  .viminfo
adavtyan@artur-lpt:~$
```

Long Display Listing:

```
adavtyan@artur-lpt:~$ sudo ls -l /root/
total 4
drwxr-xr-x 6 root root 4096 Φщη 11 03:23 snap
adavtyan@artur-lpt:~$
```

Human-Readable Size:

```
adavtyan@artur-lpt:~$ sudo ls -lh /root/
total 4.0K
drwxr-xr-x 6 root root 4.0K Φщη 11 03:23 snap
adavtyan@artur-lpt:~$
```

Sort a Listing:

```
adavtyan@artur-lpt:~$ ls -lS /etc/ssh
total 8
drwxr-xr-x 2 root root 4096 Uwj 29  2020 ssh_config.d
-rw-r--r-- 1 root root 1603 Uwj 29  2020 ssh_config
adavtyan@artur-lpt:~$
```

sourcemind

Managing Files and Directories

Copying Files: The cp command is used to copy files. It requires a source and a destination. The structure of the command is as follows:

```
cp [options] [source] [destination]
```

The source is the file to be copied. The destination is where the copy is to be located. When successful, the cp command does not have any output (no news is good news). The following command copies the /etc/hosts file to your home directory:

```
adavtyan@artur-lpt:~$ cp /etc/hosts ~
adavtyan@artur-lpt:~$ ls -l hosts
-rw-r--r-- 1 adavtyan adavtyan 259 Ապր 12 12:08 hosts
adavtyan@artur-lpt:~$
```

Verbose mode: The -v option causes the cp command to produce output if successful. The -v option stands for verbose:

```
adavtyan@artur-lpt:~$ cp -v /etc/hosts ~
'/etc/hosts' -> '/home/adavtyan/hosts'
adavtyan@artur-lpt:~$ cp -v /etc/hosts ~/hosts.copy
'/etc/hosts' -> '/home/adavtyan/hosts.copy'
adavtyan@artur-lpt:~$
```

Copying Directories: The cp command is used to copy Folders. In this example copy /tmp/conf/ folder and all its files to /tmp/backup/ directory:

```
adavtyan@artur-lpt:~$ cp -avr /tmp/conf/ /tmp/backup/
'/tmp/conf/' -> '/tmp/backup/'
adavtyan@artur-lpt:~$
```

Where,

- -a: Preserve the specified attributes such as directory an file mode, ownership, timestamps, if possible additional attributes: context, links, xattr, all.
- -r: Copy directories recursively.
- -v: Verbose output.

sourcemind

Moving Files

To move a file, use the mv command. The syntax for the mv command is much like the cp command:

```
mv [options] [source] [destination]
```

The Source can be one, or more files or directories, and Destination can be a single file or directory.

- When multiple files or directories are given as a Source, the Destination must be a directory. In this case, the Source files are moved to the target directory.
- If you specify a single file as Source, and the Destination target is an existing directory, then the file is moved to the specified directory.
- If you specify a single file as Source, and a single file as Destination target then you're renaming the file.
- When the Source is a directory and Destination doesn't exist, Source will be renamed to Destination. Otherwise if Destination exist, it be moved inside the Destination directory.

To move a file or directory, you need to have write permissions on both Source and Destination. Otherwise, you will receive a permission denied error.

```
adavtyan@artur-lpt:~$ ls hosts
hosts
adavtyan@artur-lpt:~$ mv hosts Videos/
adavtyan@artur-lpt:~$ ls hosts
ls: cannot access 'hosts': No such file or directory
adavtyan@artur-lpt:~$ ls Videos/hosts
Videos/hosts
```

Rename Files:

```
adavtyan@artur-lpt:~$ mv Videos/hosts Videos/hosts.copy
adavtyan@artur-lpt:~$ ls  Videos/hosts
ls: cannot access 'Videos/hosts': No such file or directory
adavtyan@artur-lpt:~$ ls  Videos/hosts.copy
Videos/hosts.copy
```

sourcemind

Creating and Removing Files

There are several ways of creating a new file, including using a program designed to edit a file (a text editor). There is also a way to create an empty file that can be populated with data at a later time. This feature is useful for some operating systems as the very existence of a file could alter how a command or service works. It is also useful to create a file as a "placeholder" to remind you to create the file contents at a later time.

To create an empty file, use the touch command as demonstrated below:

```
adavtyan@artur-lpt:~$ touch doc.txt
adavtyan@artur-lpt:~$ ls -l doc.txt
-rw-rw-r-- 1 adavtyan adavtyan 0 Ապր 12 13:01 doc.txt
adavtyan@artur-lpt:~$
```

Removing Files:

```
adavtyan@artur-lpt:~$ ls -l doc.txt
-rw-rw-r-- 1 adavtyan adavtyan 0 Ապր 12 13:01 doc.txt
adavtyan@artur-lpt:~$ rm doc.txt
adavtyan@artur-lpt:~$ ls -l doc.txt
ls: cannot access 'doc.txt': No such file or directory
adavtyan@artur-lpt:~$
```

As a precaution, users should use the -i option when deleting multiple files:

```
adavtyan@artur-lpt:~$ rm -i hosts.copy
rm: remove regular file 'hosts.copy'? y
adavtyan@artur-lpt:~$ ls -l hosts.copy
ls: cannot access 'hosts.copy': No such file or directory
adavtyan@artur-lpt:~$
```

sourcemind

Creating and Removing Directories

Creating Directories: To create a directory, use the mkdir command:

```
adavtyan@artur-lpt:~/Documents/Files$ ls -la
total 12
drwxrwxr-x  3 adavtyan adavtyan 4096 Ապ 12 15:04 .
drwxr-xr-x 10 adavtyan adavtyan 4096 Ապ 12 15:04 ..
drwxrwxr-x  2 adavtyan adavtyan 4096 Ապ 12 15:04 test
adavtyan@artur-lpt:~/Documents/Files$
```

Removing Directores: You can delete directories using the rm command. However, the default behavior (no options) of the rm command is to not delete directories:

```
adavtyan@artur-lpt:~/Documents$ rm Files/
rm: cannot remove 'Files/': Is a directory
adavtyan@artur-lpt:~/Documents$
```

To delete a directory with the rm command, use the -r recursive option:

```
adavtyan@artur-lpt:~/Documents$ rm -r Files/
adavtyan@artur-lpt:~/Documents$ ls Files
ls: cannot access 'Files': No such file or directory
```

You can also delete a directory with the rmdir command, but only if the directory is empty.

```
adavtyan@artur-lpt:~$ rmdir Documents/
rmdir: failed to remove 'Documents/': Directory not empty
adavtyan@artur-lpt:~$
```

NOTE: When a user deletes a directory, all of the files and subdirectories are deleted without any interactive question. It is best to use the -i option with the rm command

sourcemind

Thank you for your attention !

Q&A