Written by: justin

Score: 1033
votes: 1060
Format: Article

The 5-Minute Essential Shell Tutorial

Alright, far too often (especially in the IRC channels) there is a time where even the most beginner of users are faced with the terminal. It has many names: terminal, shell, console, "command prompt" even as a carryover from those familiar with Windows. Many people are frightened by it for some reason or another, so this tutorial will attempt to provide you the most basic of commands to enable navigation and basic system actions from the comfort of your keyboard.

Let's get started shall we? Since everyone's Mint version can be different, I'm not going to detail how to actually open the terminal. I'll assume you can find it in the menu or by right-clicking in the desktop.

Facts:

1. You can do almost anything in a terminal which you would also do from a GUI interface.

2. Most commands were designed first to work in the terminal, then a GUI put on top of them. That's why some GUI's may feel clunky - they were an afterthought at times.

3. The default location for your terminal to open from the menu is in your home folder, also known as ~

4. Your current directory can be noted by the **.** operator. Most commands when they act on the current folder selection, operate on **.**

5. Commands, locations, and files are case sensitive. /home is not the same as /HOME or /Home.

6. Use the tab key to complete file names. If you have a long driver titled, for example, driver-128947232jaseu.sh, simply type dri and it will fill in the rest, provided you don't have 2 names starting with "dri" and if you do, add another character to make it "driv" and try again.

7. Almost any command can be read about in full using the manpage or by typing -h or --help after writing the initial command. This syntax is either **man command_name**, **command_name -h**, or **command_name --help**.

8. To get even more information, you can use **info**. A command can be searched for by using **info command_name**. For most of these commands which are part of the coreutils package, one can find info as well using **info coreutils command_name invocation** where **command_name** is replaced by the command searched for.

9. Almost any command can also explicitly display what is happening. This is done usually by the **-v** or **--verbose**

10. You can specify multiple command flags to a command at a time to get more information (see the **ls -al** example below.)

11. Command names are not always obtuse - due to space limitations in the old days of Unix they were shortened, and the conventions stuck.

Commands:

cd -> Used to navigate the directories. You can move to any location by path.

1. **cd** This will move you back to your home, same as cd ~
2. **cd ..** This will take you back exactly one directory. Starting in /home/justin/Desktop, cd .. will put me into /home/justin. This can be expanded upon, cd ../../ from the Desktop location instead will move me 2 back, from my Desktop to /home.
3. **cd foldername/** This will move you forward to the given folder *in your current folder.* Take note of the missing prefix / it is an important omission. if I am in /home/justin and I want to get to Desktop, I must type cd Desktop/ without the / before Desktop. Typing / before it places us in the root of file system, which is incorrect.
4. **cd /some/other/path** This will take you to the specified folder path, supposing it exists as typed exactly. Don't forget your tab completion!

ls -> Used to list folder contents. You can view many kinds of file and folder attributes.

1. **ls** By itself, ls will simply list all your files in the current folder. From fact #4, this literally does **ls .**
2. **ls -l** Provides a longer listing format including owners, permissions, size, and date modified.
3. **ls -a** Displays hidden files and folders as well as the normal listing.
4. **ls -al** Combine options to display both hidden files and in the long format.
5. **ls -h** Show file sizes in human readable format (K, M, Gbyte) filesizes instead of bytes. Often used in conjuction with the **-l** flag.
6. You can view files in directories you are not even in. If I am in /home/justin/Desktop, and I want to view a file in /home/justin, I can do **ls ../** list files one directory back (and not have to go back to do so.)

cp -> Copy files

1. **cp file /path/to/folder** Copies specified file to the given path.
2. **cp -r folder /path/to/folder** Copies recursively the contents of the folder to another folder.
3. **cp *.extension /path/to/folder** Copies files matching the given extension to the new folder. To copy all .doc files, it becomes **cp *.doc /path/to/folder** and the folder must exist.
4. **cp name* /path/to/folder** Copies all files starting with 'name' to the given folder. To copy all files starting with example, it becomes **cp example* /path/to/folder** and the folder must exist.

mv -> Move files

1. The syntax of **mv** is similar to the example above with **cp** exempt for example #2. **mv** does not take the **-r** flag since moving a folder also moves its contents. The syntax is not exact in all instances, but works with the above examples. Consult your manpages for more details.

rm -> Remove files

1. For all intents and purposes, removing files via **rm** is permanent. It does not use the Trash bin. Use with caution and make sure you are deleting explicitly what you want, not what you think you want. If you decide to get fancy with your delete commands, it's probably going to come back to bite you.
2. **rm file** Remove the specified file from the system**.**
3. **rm -r folder** Remove the specified folder from the system
4. **rm -rf folder** Removes the specified folder **forcefully** from the system. This command can severely break your configuration if used incorrectly as it will not prompt you if something critical is being deleted. If you have to use this, chances are something more is broken or there was a mistake made. **This should only be used as an absolute last resort method and is not recommended.**

nano -> full command line text editor

1. One can edit files using **nano** in a terminal to do quick and dirty files all the way up to full configurations. It's handy, but keep in mind it handles plain text files and programming files, things like MS Word documents will not open properly!
2. If a file is owned by root, it is not editable as a normal user. **nano** must be prefixed with **sudo** in order to save changes. Otherwise, it will open in read-only mode.
3. **nano newfile.whatever** Nano creates a new file of the specified name and opens it for editing.

4. **nano existing_file** Nano opens the existing file for editing.
5. From inside **nano**
    1. **Save file** using the ctrl+o key combination, and either change the name or press entier to keep the same name. This will save the file.
    2. **Exit nano** by using ctrl+x key combination. If you have unsaved changes, it will ask if you want to save.

mkdir -> Create directories

1. **mkdir folder_name** Creates the folder with the specified name
2. **mkdir -p /path/to/folder/name** Creates each folder as necessary. To create folder /home/justin/newfolder/2ndfolder, and only /home/justin exists, using **mkdir -p** will make both directories newfolder and 2ndfolder.

ps -> List processes

1. **ps aux** List all processes in detail running on the system, including user, Process ID (PID), and name of process. Using this, one can view their process list and if necessary, kill unnecessary or stalled processes.

kill / killall / xkill -> Kill offending processes.

1. **kill PID** PID is a number referencing the offending process. One should obtain the PID from a command like **ps aux**. If a process refuses to die, one can alternatively specify **kill -9 PID** which should terminate the process by any means, even uncleanly or if it will mess up the system.
2. **killall program** Killall kills *by name* all instances of said program. If there are for example 3 firefox sessions open, **killall firefox** will do just that; kill all firefox sessions. **kill** would simply take the specified PID of the offending firefox process you wish to kill, and kill that one only.
3. **xkill** is a GUI way to click and kill windows. Typing in **xkill** should present a skull-and-crossbones icon, and the next window clicked on will be killed.

Pipes -> The most useful thing you will learn in *NIX. Redirecting output of a program to anothers input.

1. **Pipes** are represented by the ' straight bar ' otherwise known as the ' | ' key.
2. It is a rarely used key in Windows, it is often found on the backslash key.
3. They are used to link commands together. **Pipes** take the output of one command and route it to be used as input for a second command chained together.
4. Consult more online resources with information about **pipes** and their use as there are volumes.

> and >> redirectors -> Send output to a file instead of the terminal.

1. > is used to *overwrite* currently existing files contents and replace with the output from the new command.
2. >> is used to *append* information to currently existing files. This is useful for logging.
3. Example: **ps aux > processes.log** Sends the output of **ps aux** to the file **processes.log** for viewing the command output in a text editor and overwrites the current contents of the file.

tee -> Send output to both a file and the terminal

1. **tee** is used in conjunction with a ' | ' in order to take the command output and send it elsewhere. This is useful if there are errors which fly by the screen before you can read them, this way whatever goes on the screen is also captured to a file.
2. Example: **dmesg | tee boot.txt** would run the command **dmesg** which shows the initial boot info, and the ' | ' sends the output of **dmesg** to **tee**, which then does its job by sending it to the terminal and to the log file **boot.txt**.

File Execution -> So you want to execute files or programs from the terminal? Make sure it's marked executable. If not, see Quick Tip #4 below.

1. Need to execute a file in the current directory after it is marked executable? The **./** operator can execute the file as a normal user provided you do not need root rights. **./** literally means "in the current directory" so it does not work on files outside of the present directory.

2. Need to execute a file *not* in the current directory? You must pass the path to the proper executing program. If it is a python program, it's **python /path/to/file** and if it is a shell file, it is **sh /path/to/file** as an example. There are of course other programs, but these will be the most common for

beginners.

3. Need to execute a file with root rights because you received operation not permitted? Prefix the command with **sudo**. Thus, from the above example, **sudo python /path/to/file** will execute the script with root rights.

4. Need to execute a GUI program from the terminal? Simply type the program name (case sensitive!) and it will launch. This will render the current terminal unusable. Closing the terminal while the program is open will kill the program. A better way is to background the program, using **program_name &** and then typing the word **exit** at the terminal to close it and keep the process running.

5. Need to run a GUI program with root rights from the terminal? Prefix it with gksudo or gksu and **not sudo**. Using sudo to launch GUI applications is a bad habit and should be avoided.

6. **Do not, do *not* use sudo simply because something receives "Operation not permitted." Keep in mind what you are doing as you can absolutely *destroy* systems by running commands in the wrong place with root rights. This point cannot be emphasized enough. Make sure your files come from reputable sources**.

Quick tips:

1. Lost yourself in a directory? Not sure where you are? Type **pwd** to **p**rint **w**orking **d**irectory.

2. Want to calculate your disk space quickly? **df -h** can give you a quick checkup.

3. Want to calculate the size of a folder or file quickly? **du -cksh target_name** can do exactly that. Want to calculate the size of the current folder? **du -cksh .**

4. Need to mark a file executable? **chmod +x filename** can do that. Next time you see a file you need to execute and it is not marked executable, now you know how to fix it.

5. Want to mount an iso like Daemon-Tools on Windows? Linux has this functionality built in. Simply create a directory somewhere, say /home/justin/isomount, and issue the command **mount -o loop /path/to/myisofile.iso /home/justin/isomount** and the contents will be mounted inside that folder.

6. Run a command before, you need to re-run it, but you can't really remember what it was exactly? Type **history** into the terminal and it will print out your command history. Want to clear your history? **history -c** will wipe the information.

Tags: *bash, shell, commands, terminal*
Created: 6 years ago.
Last edited: 6 years ago.
Read 72857 times.