**Section 1:**

1. What Is Linux?

Linux is an [open source](https://www.redhat.com/en/topics/open-source/what-is-open-source) operating system (OS). An [operating system](https://www.redhat.com/en/technologies/linux-platforms/old-enterprise-linux) is the software that directly manages a system’s hardware and resources, like CPU, memory, and [storage](https://www.redhat.com/en/topics/data-storage/software-defined-storage).

1. What is Linux Kernel?

The Linux kernel is the main component of a [Linux operating system (OS)](https://www.redhat.com/en/topics/linux/what-is-linux) and is the core interface between a computer’s hardware and its processes. It communicates between the 2, managing resources as efficiently as possible. It exists within the OS and controls all the major functions of the hardware.

1. What is Linux Shell?

A shell is special user program which provides an interface to user to use operating system services.

1. Unix philosophy.

The **Unix philosophy**, originated by [Ken Thompson](https://en.wikipedia.org/wiki/Ken_Thompson), is a set of cultural norms and philosophical approaches to [minimalist](https://en.wikipedia.org/wiki/Minimalism_(computing)), [modular](https://en.wikipedia.org/wiki/Modularity_(programming)) [software development](https://en.wikipedia.org/wiki/Software_development). It is based on the experience of leading developers of the [Unix](https://en.wikipedia.org/wiki/Unix) [operating system](https://en.wikipedia.org/wiki/Operating_system). Early Unix developers were important in bringing the concepts of modularity and reusability into software engineering practice, spawning a "[software tools](https://en.wikipedia.org/wiki/Software_tools)" movement. Over time, the leading developers of Unix (and programs that ran on it) established a set of cultural norms for developing software; these norms became as important and influential as the technology of Unix itself and have been termed the "Unix philosophy."

1. The role of shells in the Linux environment?

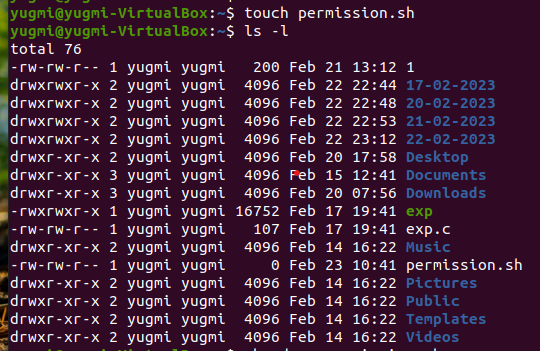
Shell accepts human readable commands from user and convert them into something which kernel can understand. It is a command language interpreter that execute commands read from input devices such as keyboards or from files. The shell gets started when the user logs in or start the terminal.

1. Shebang.

The shebang is the combination of the # (pound key) and ! (exclamation mark). This character combination has a special meaning when it is used in the very first line of the script. It is used to specify the interpreter with which the given script will be run by default. This #! is called shebang or hashbang.

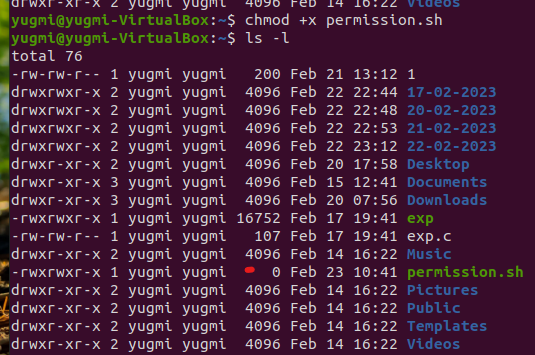
1. Setting up permissions on a script & Execute, Debug.

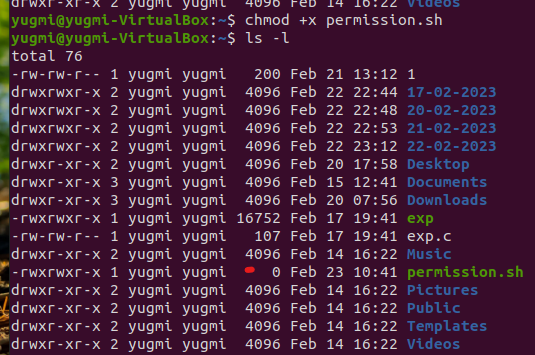
The [chmod command](https://bash.cyberciti.biz/guide/Chmod_command) (change mode) is a shell command in Linux. It can change file system modes of files and directories. The modes include permissions and special modes.

****

**Graphical user interface, text

Description automatically generated**

****

****

**Graphical user interface, text

Description automatically generated**

**Graphical user interface, text

Description automatically generated**

**Graphical user interface, text

Description automatically generated**

**Graphical user interface, text

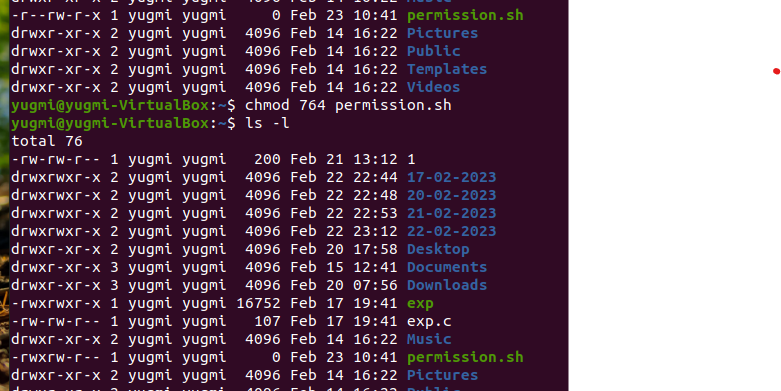
Description automatically generated**

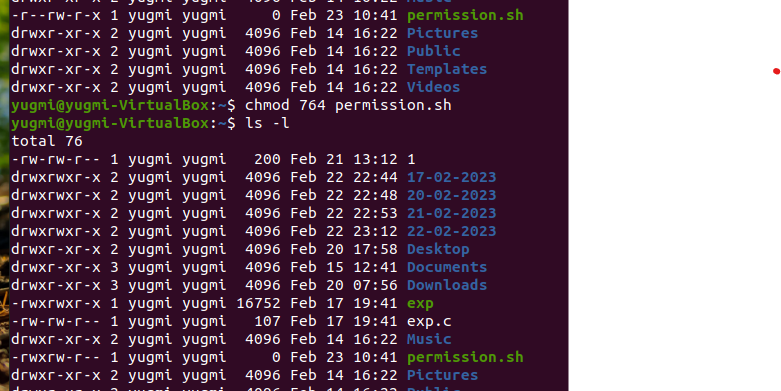
**Text

Description automatically generated with medium confidence**

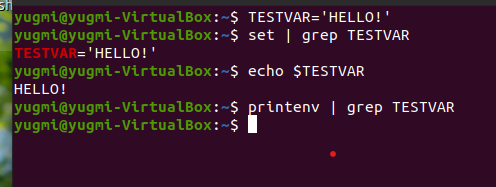
**Text

Description automatically generated with medium confidence**

****

****

1. Display the value of shell variables.



Here, printenv will not show any output that means it is not environment variable.

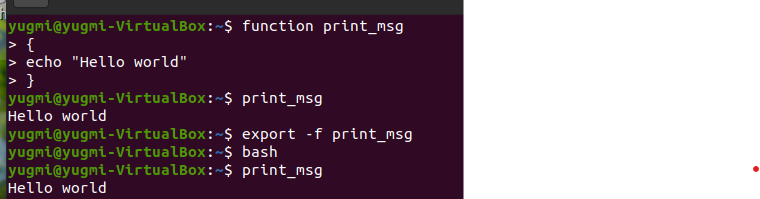
1. The export statement.

The export command is a built-in [Bash](https://phoenixnap.com/kb/bash-function) shell command that exports [environmental variables](https://phoenixnap.com/kb/linux-set-environment-variable) as child processes without affecting the existing environment variables.

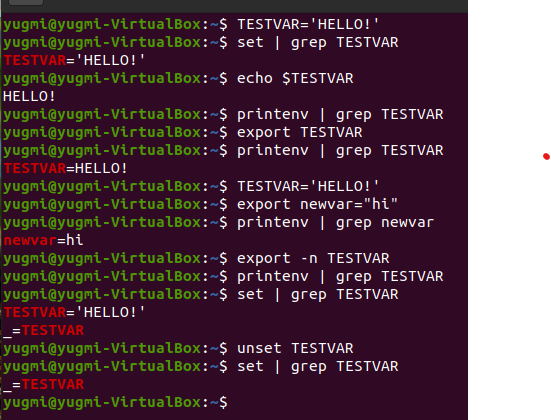
Linux export options:

| **Option** | **Description** |
| --- | --- |
| **-f** | Exports **[name]**s as functions. |
| **-n** | Allows users to remove **[name]**s from the list of exported variables. |
| **-p** | Displays a list of all exported variables and functions in the current shell. |

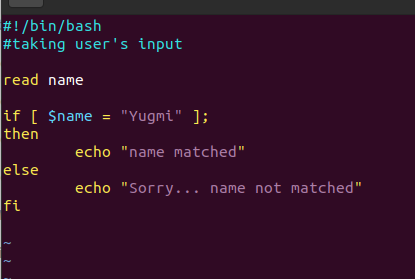
Text

Description automatically generated 

1. Unset shell and environment variables.



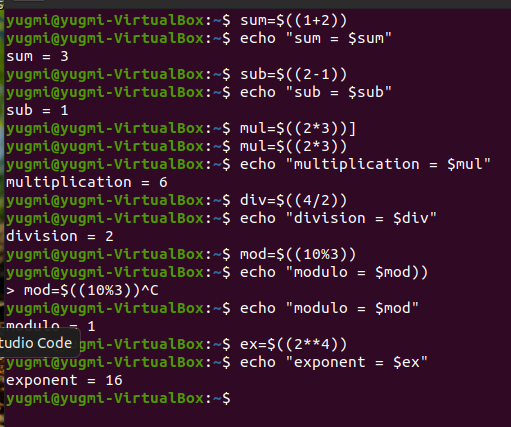
1. Getting User Input Via Keyboard.



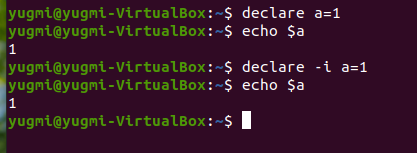
Text

Description automatically generated

1. Perform arithmetic operations.



1. Create an integer variable.



1. Create the constants variable.

Text

Description automatically generated

1. Bash variable existence check.

Text

Description automatically generated

A picture containing text

Description automatically generated

1. Recalling command history.

Text

Description automatically generated

1. Path name expansion.

Text

Description automatically generated

1. Create and use aliases.

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

1. The tilde expansion.

Graphical user interface, text

Description automatically generated

1. Using aliases.

Text

Description automatically generated

1. Test command.

**Flags for files and directories:**

* **test -e filename:** Checks whether the file exists or not. And return 1 if file exists and returns 0 if file does not exist.
* **test -d filename:** Checks whether the file is a directory or not. And returns 0 if the file is a directory and returns 1 if the file is not a directory.
* **test -f filename:**Checks whether the file is a regular file or not. And returns 0 if the file is a regular file and returns 1 if the file is not a regular file.
* **test -s filename:** Checks whether the file is empty or not. And returns 0 if the file is not empty and returns 1 if the file is empty.
* **test -r filename:** Checks whether the file is readable or not. And returns 0 if the file is readable and returns 1 if the file is not readable.
* **test -w filename:** Checks whether the file is writable or not. And returns 0 if the file is writable and returns 1 if the file is not writable.
* **test -x filename:**Checks whether the file is executable or not. And returns 0 if the file is executable and returns 1 if the file is not executable.

A picture containing text

Description automatically generated

**Section 2:**

1. If structures to execute code based on a condition.

Text

Description automatically generated

**Text

Description automatically generated**

1. If..else..fi

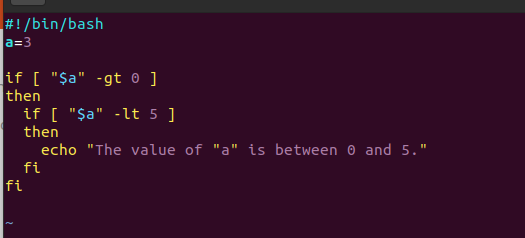
Text

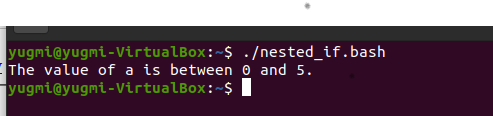
Description automatically generated

Text

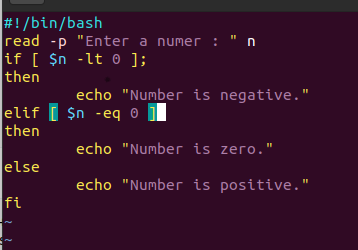
Description automatically generated

1. Nested ifs.





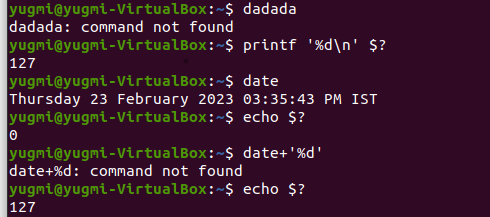
1. Multilevel if-then-else.



Text

Description automatically generated

1. The exit status of a command.

 Every Linux or Unix command executed by the shell script or user has an exit status. Exit status is an integer number. 0 exit status means the command was successful without any errors. A non-zero (1-255 values) exit status means command was a failure.

1. Conditional execution.

Conditional execution can be done using && and || .

Text

Description automatically generated

1. Numeric comparison.

Text

Description automatically generated

**Text

Description automatically generated**

1. String comparison.

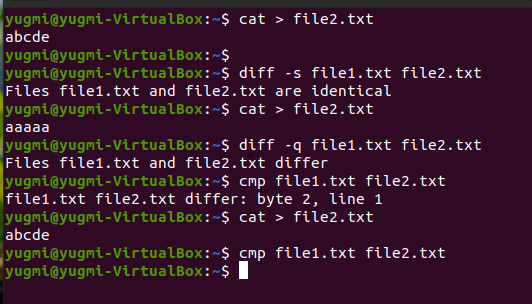
Text

Description automatically generated

Text

Description automatically generated

1. File attributes comparisons.

****

1. Shell command line parameters.

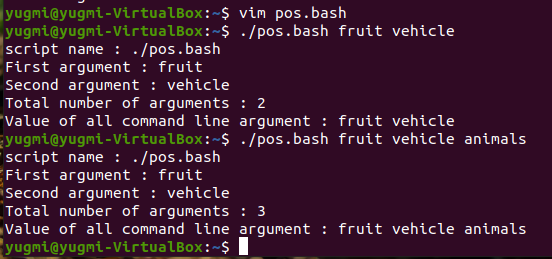
Command line parameters are a way to pass information into a program or script in order for it to do what you want it to. Some examples of command line parameters:

$ ls -l

$ cat file.txt

1. How to use positional parameters.

All command line parameters ( positional parameters ) are available via special shell variable $1, $2, $3,...,$9.

****

1. Parameters Set by the Shell.

Bash shell set several special parameters.

All command line parameters or arguments can be accessed via $1, $2, $3,..., $9.

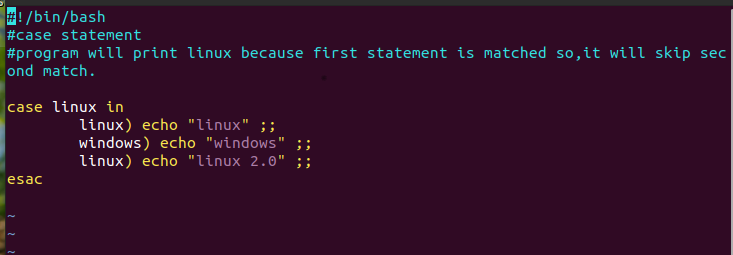
* [**$\***](https://bash.cyberciti.biz/guide/$*) holds all command line parameters or arguments.
* [**$#**](https://bash.cyberciti.biz/guide/$) holds the number of positional parameters.
* [**$-**](https://bash.cyberciti.biz/guide/$-) holds flags supplied to the shell.
* [**$?**](https://bash.cyberciti.biz/guide/$%3F) holds the return value set by the previously executed command.
* [**$$**](https://bash.cyberciti.biz/guide/$$) holds the process number of the shell (current shell).
* [**$!**](https://bash.cyberciti.biz/guide/$!) hold the process number of the last background command.
* [**$@**](https://bash.cyberciti.biz/guide/$@) holds all command line parameters or arguments.

1. Create usage messages.
2. Exit command.

exit command in linux is used to exit the shell where it is currently running. It takes one more parameter as *[N]* and exits the shell with a return of status *N*. If n is not provided, then it simply returns the status of last command that is executed. Syntax:

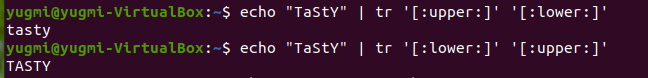
**exit [n]**

1. The case statement.





1. Dealing with case sensitive pattern.



**Section 3:**

1. The for loop statement.

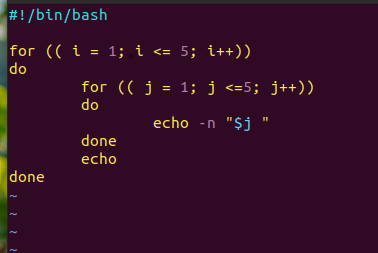
Text

Description automatically generated

A picture containing text

Description automatically generated

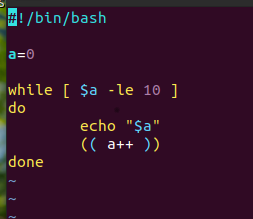
1. Nested for loop statement.



A picture containing diagram

Description automatically generated

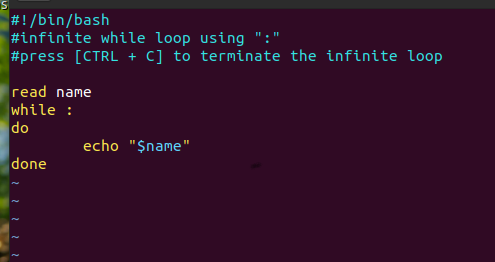
1. The while loop statement.



**Text

Description automatically generated**

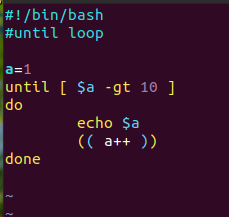
1. Use of **:** to set infinite while loop.

****

**A picture containing text

Description automatically generated**

1. The until loop statement.

****

**A screenshot of a computer

Description automatically generated with medium confidence**

1. The select loop statement.

Text

Description automatically generated

**Text

Description automatically generated**

1. Exit the select loop statement.

Text

Description automatically generated

1. Using the break statement.

Text

Description automatically generated

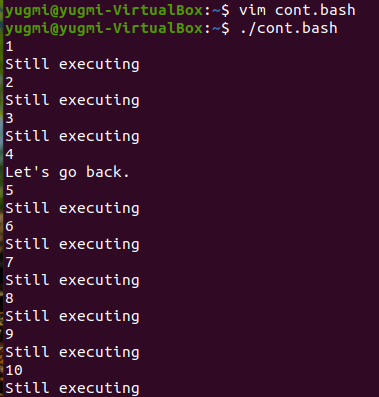
Text

Description automatically generated

1. Using the continue statement.

Text

Description automatically generated



**Section 4:**

* 1. Command substitution.

Command substitution means nothing more but to run a shell command and store its output to a variable or display back using [**echo command**](https://bash.cyberciti.biz/guide/Echo_command). For example, display date and time:

echo "Today is $(date)"

You can use the grave accent (`) to perform a command substitution. The syntax is:

`command-name`

OR

$(command-name)

Text

Description automatically generated



* 1. Input and Output.

Almost all commands produce the output to screen or take input from the keyboard, but in Linux it is possible to send output to a file or to read input from a file. Each shell command has its own input and outputs. Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell.

* 1. Standard input.

Standard input is a term for the input that a command-based program receives. In interactive use, it is normally from the keyboard, but it can also come from a file.

* 1. Standard output.

Standard output, like standard input, is where a program will send its text output. This is typically a terminal emulator on modern systems but in the past was also on physical terminals, either with CRT screens or printed on paper using teletypes.

Like standard input, you can also redirect standard output to a file.

* 1. Standard error.

Standard error is usually used for any error messages a program may generate. As with standard output, it is usually displayed on the screen but can also be redirected to a file or to a block device like **/dev/null**.

* 1. Empty file creation.

The easiest way to create a new file in Linux is by [using the touch command](https://phoenixnap.com/kb/touch-command-in-linux).

Syntax:

touch test.txt

* 1. Redirection of standard error.

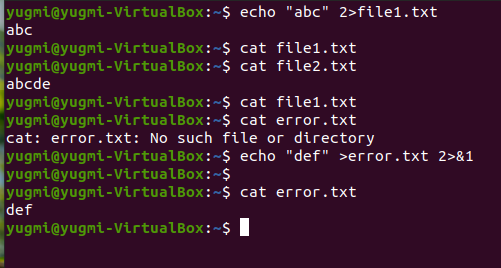
The redirection operator (command > file) only redirects standard output and hence, the standard error is still displayed on the terminal. The default standard error is the screen. The standard error can also be redirected so that error messages do not clutter up the output of the program. ‘2’ denotes the stderr of a program.

**2> stderr**

‘2>’ redirects the error of an output to a file. The error messages are redirected and clean output is displayed.

**2>&1**

When 2>& is used both standard error and standard output get redirected to the same file.

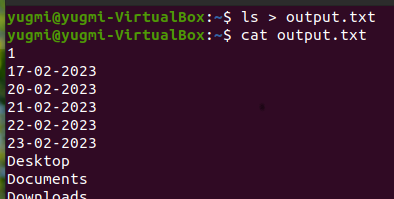


* 1. Redirection of standard output.

simplest form of redirection is output redirection also called stdout redirection.

With output redirection, you can redirect the output to a file. If this output files doesn’t exist, the shell will create it.

The **‘>**‘ symbol is used for output (STDOUT) redirection.

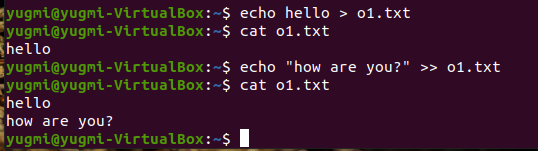


Graphical user interface

Description automatically generated with medium confidence

* 1. Appending redirected output.

We use “>>” to append redirected output.



* 1. Redirection of both standard error and output.

**2>&1**

When 2>& is used both standard error and standard output get redirected to the same file.

Graphical user interface, text

Description automatically generated

* 1. Writing output to files.

The tee command send the output to the display as well as to a file (or as input to another command). You can use it like this:

command | tee file.txt

Calendar

Description automatically generated with medium confidence

* 1. Assigns the file descriptor (fd) to file for output
  2. Assigns the file descriptor (fd) to file for input
  3. Closes the file descriptor (fd)
  4. Opening the file descriptors for reading and writing
  5. Reads from the file descriptor (fd)
  6. Executes commands and send output to

**Section 5:**

* 1. Linking Commands.

‘ln’ is a command-line utility for creating links between files. By default, the ln command creates hard links. To create a symbolic link, use the -s ( --symbolic ) option.

To overwrite the destination path of the symlink, use the -f (--force) option.

**ln -sf my\_file.txt my\_link.txt**

Text

Description automatically generated

To delete a symlink, invoke the **rm** and **unlink** command followed by the symbolic link name is used.



* 1. Multiple commands.

On Linux, there are three ways to run multiple commands in a terminal:

1. **The Semicolon (;) operator:**

When there’s a need to run two or more related terminals commands such that the output status of the first command doesn’t affect the execution of the latter, the semicolon operator is the way to go.

1. **The Logical OR (||) operator:**

We can use the OR operator when we need to run two related commands together such that the shell executes the next command only when the previous one fails.

1. **The Logical AND (&&) operator:**

 the AND operator executes the next command in a sequence only when its previous command runs successfully.

Text

Description automatically generated

* 1. Putting jobs in background.

To run a command in the background, add the ampersand symbol (&) at the end of the command.

**command &**

* 1. Pipes.
* In Linux, the pipe command lets you sends the output of one command to another. Piping, as the term suggests, can redirect the standard output, input, or error of one process to another for further processing.
* The syntax for the pipe or unnamed pipe command is the “ | ” character between any two commands.

Command-1 | Command-2 | …| Command-N

1. How to use pipes to connect programs.

You redirect the standard output of the last command using a pipe with ‘ [< ‘ redirection](https://bash.cyberciti.biz/guide/Standard_output) symbol.

Text

Description automatically generated

1. Input redirection in pipes.

Text

Description automatically generated

1. Output redirection in pipes.

You redirect the standard output of the last command using a pipe with [> or >> redirection](https://bash.cyberciti.biz/guide/Standard_output) symbol.

Sort all process memory wise and save the output to a file.

Text

Description automatically generated

To copy files of 1 directory to other.

cp -r dir1/\* dir2/

For new tab in terminal:

Ctrl+shift+t