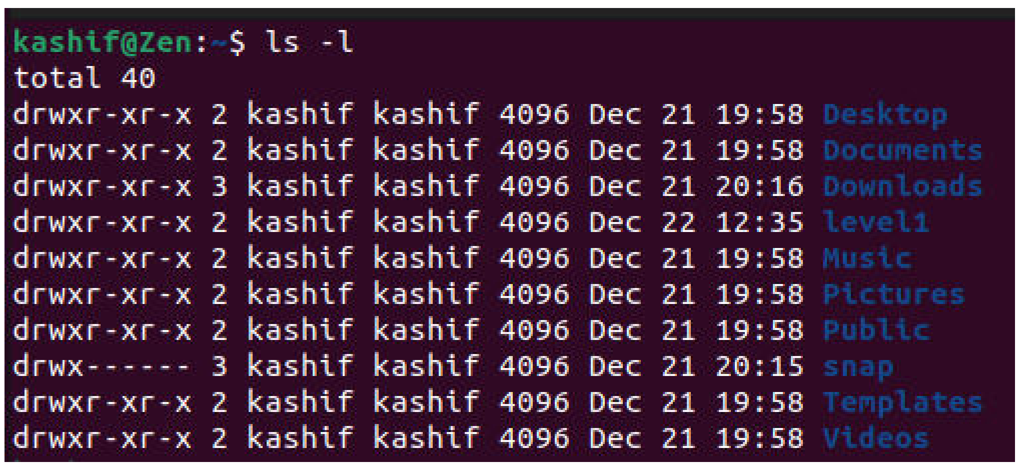
Simple Commands in Linux - 2

Type	📒 Lecture
Date	@December 22, 2021
Lecture #	1
Lecture URL	https://youtu.be/lpe6AKk5GIk
Notion URL	https://21f1003586.notion.site/Simple-Commands-in-Linux-2-0cd2a51884c14be1a43ad24cd99c6692
Week #	2

`ls`

- Short and Long form of options
- Interpretation of directory as an argument
- Recursive listing
- Order of options on command line

  `ls -l` **will display the files & folders in the current directly, in a list**



```
kashif@Zen:~$ ls -l
total 40
drwxr-xr-x 2 kashif kashif 4096 Dec 21 19:58 Desktop
drwxr-xr-x 2 kashif kashif 4096 Dec 21 19:58 Documents
drwxr-xr-x 3 kashif kashif 4096 Dec 21 20:16 Downloads
drwxr-xr-x 2 kashif kashif 4096 Dec 22 12:35 level1
drwxr-xr-x 2 kashif kashif 4096 Dec 21 19:58 Music
drwxr-xr-x 2 kashif kashif 4096 Dec 21 19:58 Pictures
drwxr-xr-x 2 kashif kashif 4096 Dec 21 19:58 Public
drwx------ 3 kashif kashif 4096 Dec 21 20:15 snap
drwxr-xr-x 2 kashif kashif 4096 Dec 21 19:58 Templates
drwxr-xr-x 2 kashif kashif 4096 Dec 21 19:58 Videos
```

  If we add the name of a folder after `ls -l`, it will display the files & folders of that folder

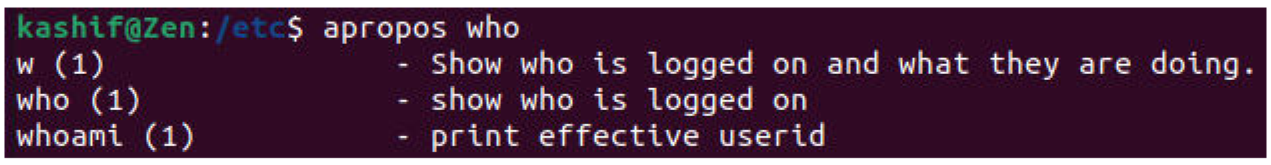
Commands to work with text files

- `less <file-name>` → To read text files, page-by-page
  - Press `q` to exit
- `cat <file-name>` → To read the entire text file, and dump them on the terminal window

    - Depending on the situation, `less` can be a better way to display the contents of a file on the terminal window rather than `cat`

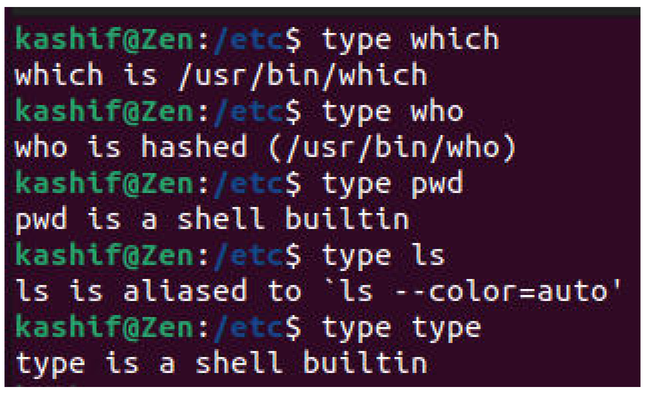
- `more <file-name>` → Works similarly to `less`
    - View contents of the file page-by-page
- `head <file-name>` → Displays the first 10 lines of a file
    - Takes an optional flag, `-n` followed by an integer to display that number of lines
- `tail <file-name>` → Works the opposite of `head` command
- `wc <file-name>` → Displays the number of lines (or newlines), words and bytes in a file
    - Takes optional flags like `-l` to display only the # of lines
- `which <command>` → A command to locate the commands, *heh*

- `whatis <command>` → Gives a brief description of a command
- `apropos <keyword>` → Takes in the keyword and returns a list of commands that contain the keyword in their name or in their description



```
kashif@Zen:/etc$ apropos who
w (1)                - Show who is logged on and what they are doing.
who (1)              - show who is logged on
whoami (1)           - print effective userid
```

- `help` → Displays the reserved keywords
    - Can pass a command name as optional parameter and it'll display it's help
    - Works with `shell keyword`
- `info` → Open a file (which is similar to a webpage) that lists all the commands, and we can move the cursor to any command and press ***Enter*** to get the manual of that command
    - Use the `<` button to go back, how do you press the `<` button? `Shift + ,`
- `type` → Know the type of a command



```
kashif@Zen:/etc$ type which
which is /usr/bin/which
kashif@Zen:/etc$ type who
who is hashed (/usr/bin/who)
kashif@Zen:/etc$ type pwd
pwd is a shell builtin
kashif@Zen:/etc$ type ls
ls is aliased to `ls --color=auto'
kashif@Zen:/etc$ type type
type is a shell builtin
```

What are aliases?

Aliases are a nickname given to a command, which may want for our convenience

How to set an alias?

```
alias <name>='<your-command>'

# Example
alias ll='ls -al'
```

How to remove an alias?

```
unalias <name>

# Example
unalias ll
```

OK, How do I know all of my aliases?

```
# Type 'alias' in the bash shell
alias
```

Multiple arguments

- Second argument
- Interpretation of last argument
- Recursion is assumed for `mv` but not for `cp`

Links

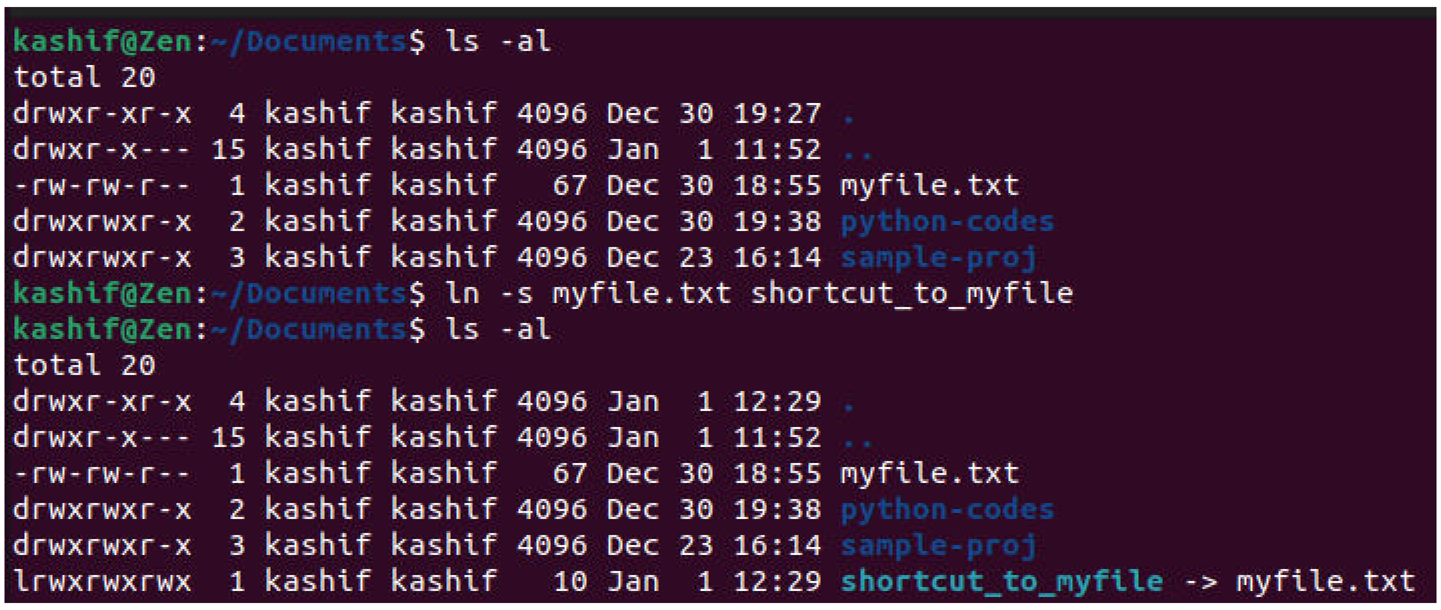
- Hard links
- Symbolic links
    - It is a bit analogous to desktop shortcuts in Windows

How to make a symbolic link?

- The command `ln` is used, with the `-s` flag

```
ln -s <source> <destination>
```

Example



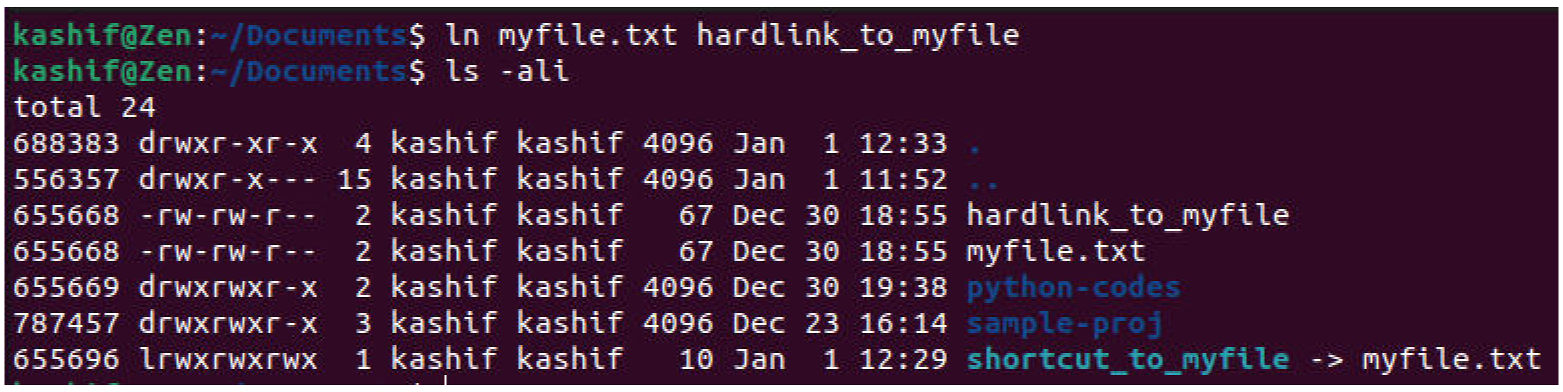
```
kashif@Zen:~/Documents$ ls -al
total 20
drwxr-xr-x  4 kashif kashif 4096 Dec 30 19:27 .
drwxr-x--- 15 kashif kashif 4096 Jan  1 11:52 ..
-rw-rw-r--  1 kashif kashif   67 Dec 30 18:55 myfile.txt
drwxrwxr-x  2 kashif kashif 4096 Dec 30 19:38 python-codes
drwxrwxr-x  3 kashif kashif 4096 Dec 23 16:14 sample-proj
kashif@Zen:~/Documents$ ln -s myfile.txt shortcut_to_myfile
kashif@Zen:~/Documents$ ls -al
total 20
drwxr-xr-x  4 kashif kashif 4096 Jan  1 12:29 .
drwxr-x--- 15 kashif kashif 4096 Jan  1 11:52 ..
-rw-rw-r--  1 kashif kashif   67 Dec 30 18:55 myfile.txt
drwxrwxr-x  2 kashif kashif 4096 Dec 30 19:38 python-codes
drwxrwxr-x  3 kashif kashif 4096 Dec 23 16:14 sample-proj
lrwxrwxrwx  1 kashif kashif   10 Jan  1 12:29 shortcut_to_myfile -> myfile.txt
```

How to create a Hard link?

Just don’t pass the `-s` flag to the `ln` command

```
ln <source> <destination>
```

Example



```
kashif@Zen:~/Documents$ ln myfile.txt hardlink_to_myfile
kashif@Zen:~/Documents$ ls -ali
total 24
688383 drwxr-xr-x  4 kashif kashif 4096 Jan  1 12:33 .
556357 drwxr-x--- 15 kashif kashif 4096 Jan  1 11:52 ..
655668 -rw-rw-r--  2 kashif kashif   67 Dec 30 18:55 hardlink_to_myfile
655668 -rw-rw-r--  2 kashif kashif   67 Dec 30 18:55 myfile.txt
655669 drwxrwxr-x  2 kashif kashif 4096 Dec 30 19:38 python-codes
787457 drwxrwxr-x  3 kashif kashif 4096 Dec 23 16:14 sample-proj
655696 lrwxrwxrwx  1 kashif kashif   10 Jan  1 12:29 shortcut_to_myfile -> myfile.txt
```

Notice the `inode` number for the hardlink is the same as the file

Size of files

- `ls -s` lists the files with file sizes

- `stat <file-name>` lists various other details as well as the file size
- `du <file-name>` shows the size of the file
  - Takes an optional `-h` flag to format the output in human readable form

In-memory filesystem

- `/proc`
- `/sys`

These directories are not stored on your HDD/SSD

They are located in your system memory

These contain the info about the system, so *no `rm -rf`-ing around as `root`*

A few commands

- `free` → To display the system memory stats, like used memory, free memory
  - Takes an optional `-h` flag to format the output in human readable form
- `df -h` → To know about the partitions, in human readable form

$hell variables

Type	📒 Lecture
Date	@December 22, 2021
Lecture #	2
Lecture URL	https://youtu.be/pPRge8Yxbso
Notion URL	https://21f1003586.notion.site/hell-variables-7c2117f90cee4aa7aec3ee410038f32a
Week #	2

`echo`

`echo` command prints a string or a environment variable

Example:

`echo $HOME`

`echo Hello, World`

Frequently used shell variables

- `$USERNAME`
- `$HOME`
- `$HOSTNAME`
- `$PWD`
- `$PATH`

Special shell variables

- `$0` → Name of the shell
- `$$` → Process ID of the shell
- `$?` → Return code of previously run program
- `$-` → Flags set in the bash shell

Process control

- Use of `&` to run a job in the background
- `fg`

- `coproc`
- `jobs`
- `top`
- `kill`

Program exit codes

- 0 → success
- 1 → failure
- 2 → misuse
- 126 → command cannot be executed
- 127 → command not found
- 130 → processes killed using `Ctrl + C`
- 137 → processes killed using `kill -9 <pid>`

Flags set in `bash`

- `h` → locate and hash commands
- `B` → brace expansion enabled
- `i` → interactive mode
- `m` → job control enabled
- `H` → ! style history substitution enabled
- `s` → commands are read from stdin
- `c` → commands are read from arguments

`echo`

```
echo Hello World
echo "How do you do?"
```

it's a start

We can enclose the string in quotes as well

Make sure to match the quotes properly

Some common `echo` commands

```
echo $USERNAME
echo $USER
echo $PWD
echo $HOME
echo $HOSTNAME
echo $PATH
```

We can enclose the shell variable (marked as $) in double quotes, and it'll replace the value

However, if we use single quote around the shell variable, the shell variable is printed as it is

We can also escape the shell variable by using a backslash in front of it

Example → `echo "User is \$USERNAME"` will display `User is $USERNAME`

To view all the shell variables

```
printenv
env
```

`printenv` prints all or part of the environment

```
printenv HOME
```

`set`

`set` allows you to change the values of shell options and set the positional parameters, or to display the names and values of shell variables. (Source)

To escape any aliases set already

```
\<alias>

# Example
\ls
\date
```

Linux Process Management

Type	📒 Lecture
Date	@December 22, 2021
Lecture #	3
Lecture URL	https://youtu.be/2aThmDRvSWU
Notion URL	https://21f1003586.notion.site/Linux-Process-Management-d73af14458004045b281f5995be7cf32
Week #	2

`sleep` command

- It is a delay for a specified amount of time
    - The time is specified in seconds

Example

```
# The following command will make the prompt sleep for 5 seconds
sleep 5
```

Coprocess

- The shell command → `coproc`
- A Coprocess is executed asynchronously in a subshell
    - In simple words, this command is used to run a process without actually losing control of the prompt
    - As we don't lose prompt access, we can execute other commands

Usage

```
coproc sleep 20
```

When the above mentioned process is completed, the prompt will output a message saying `Done`

We can run the `ps` command to view the running status of the aforementioned command

`kill`

- This command is used to kill an already running process

- *`Ctrl + C` is also quite handy*

Usage

```
kill -9 <PID>
```

How do I know the PID?

`ps` command, try `ps --forest` instead

`&` sign to put a process in the background

- We can also put the `&` (ampersand) sign at the end of a command to put that process in the background

Usage

```
sleep 30 &
```

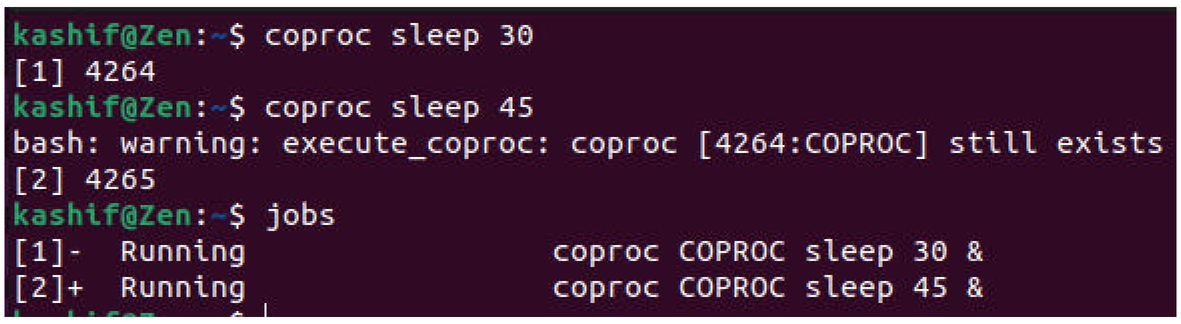
Alright, how do I bring it to the foreground?

```
fg
```

`jobs`

List the commands that are running in the background

Example



```
kashif@Zen:~$ coproc sleep 30
[1] 4264
kashif@Zen:~$ coproc sleep 45
bash: warning: execute_coproc: coproc [4264:COPROC] still exists
[2] 4265
kashif@Zen:~$ jobs
[1]-  Running                 coproc COPROC sleep 30 &
[2]+  Running                 coproc COPROC sleep 45 &
```

`top` command

It's like your Windows Task Manager, but in the Linux terminal

To exit out of `top`, press `Q` or `Ctrl + C`

Pressing `Ctrl + Z` while `top` is open suspends the process, then you can do your work and come back to it by using the `fg` command

> `jobs` command will show the top process

Well, `Ctrl + Z` will suspend any running process

$-

```
echo $-
```

The output of this shell variable would tell us about the bash shell we are currently using

To know what the output means, type `man bash` and match the flags

Launch a bash shell which is not interactive

```
bash -c "echo \$-"
```

```
bash -c "echo \$-; ps --forest;"
```

To know the PID of the new bash shell we launcher

```
bash -c "echo \$$; echo \$-; ps --forest;"
```

`history`

The following command displays out all the commands that have been run on the current shell, chronologically

```
history
```

The following command will run the `n`-th command that has been run, make sure to put `n` from the list of commands output from the above command

`n` is integer

```
!n
```

The following command will run the previous command that was executed on the bash shell

```
!!
```

Brace Expansion

So the output of `echo $-` had a `B` flag, it refers to Brace Expansion

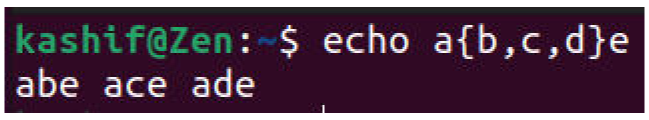
is this a JJK reference? ブレース展開

So, What is a Brace Expansion?

- Brace expansion is a mechanism by which arbitrary strings may be generated. Source

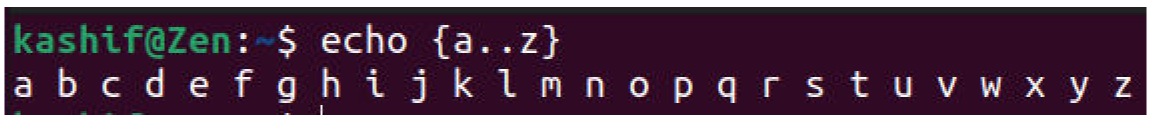
Example

```
echo a{b,c,d}e
```



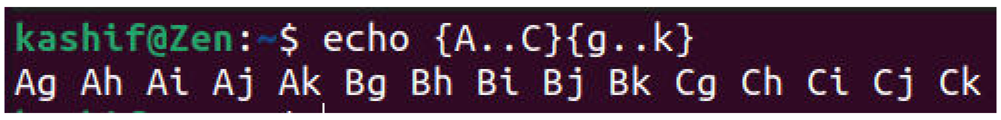
```
kashif@Zen:~$ echo a{b,c,d}e
abe ace ade
```

```
echo {a..z}
```



```
kashif@Zen:~$ echo {a..z}
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

```
echo {A..C}{g..k}
```



```
kashif@Zen:~$ echo {A..C}{g..k}
Ag Ah Ai Aj Ak Bg Bh Bi Bj Bk Cg Ch Ci Cj Ck
```

```
# The * is a wildcard character, it expands to all the files in the current dir
echo *
```

Similarly ...

```
# The following command will return all the files folders that start with D
echo D*
```

Exit codes

- `0` → All good, no error

- `1` → General errors, misc. errors
- `2` → Misuse of shell built-in
- `126` → Command invoked cannot execute
- `127` → Command not found
- `128` → Invalid argument to exit
  - `exit` takes int from `0 - 255`
- `128+n` → Fatal error signal "n"
  - When a process running in another place (like in another shell) was killed from somewhere else (killed from not the same shell)
  - `kill -9 $PPID` → `$?` returns $137\ (127 + 9)$
- `130` → Script terminated by `Ctrl + C`
- `255*` → Exit status out of range

If the exit code is out of range, bash takes the modulo of the exit code w.r.t. 256

Example

```
# Create a new bash subshell and exit with the status code 300
bash -c "exit 300"

# Check the exit code
echo $?

# The output will be ...
44
```