Combining commands and files

Type	📒 Lecture
Date	@January 9, 2022
Lecture #	1
Lecture URL	https://youtu.be/Lcx9UsS7y8Y
Notion URL	https://21f1003586.notion.site/Combining-commands-and-files-2476c1091a704743840ae8b76ab078c9
Week #	3

Executing multiple commands

- `command1; command2; command3`
  - Each command will be executed one after the other
- `command1 && command2 && command3`
  - This works as a logical AND
  - The subsequent commands after `command-n` will not run if the previous command resulted in an error
- `command1 || command2 || command3`
  - This works as a logical OR
  - The subsequent commands after `command-n` will not run if the previous command resulted in a success

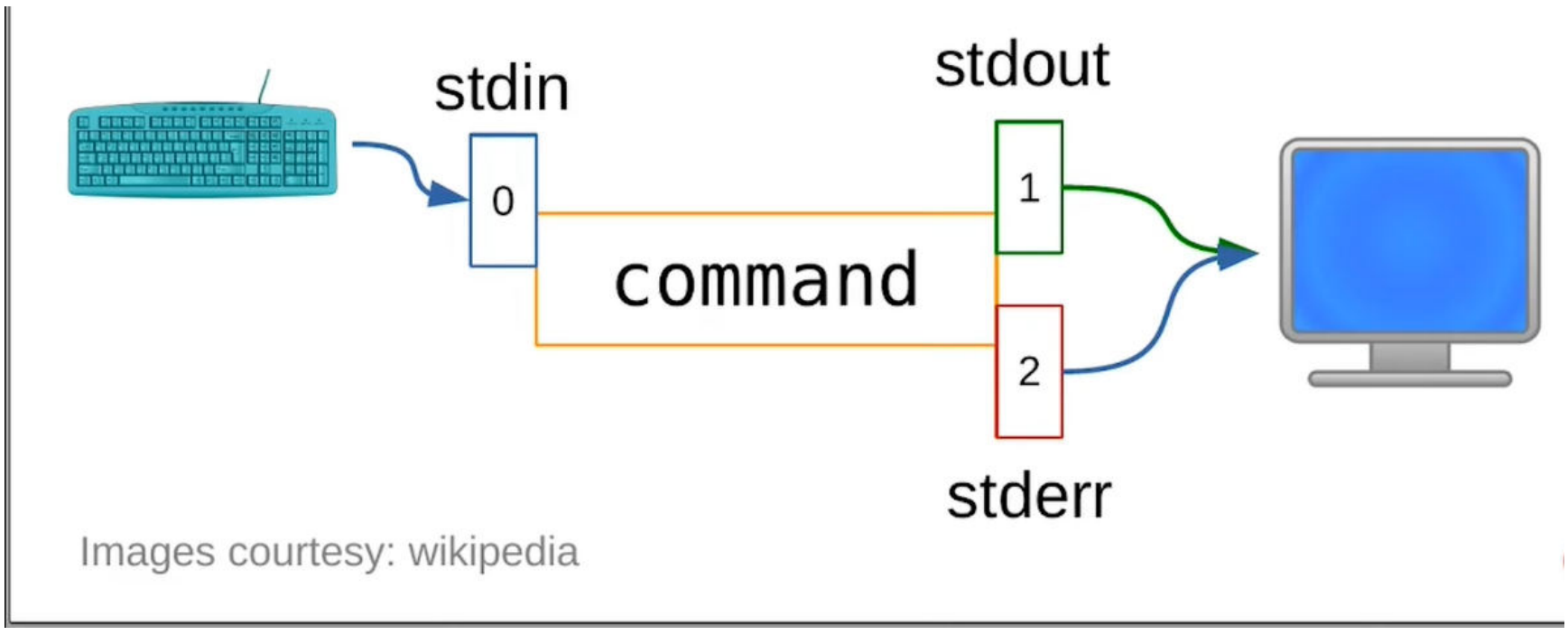
`(<command>)`

We can run any command enclosed within parentheses to execute them in a subshell, and returned back the result

We can execute a subshell within a subshell too

```
kashif@Zen:~$ echo $BASH_SUBSHELL
0
kashif@Zen:~$ (echo $BASH_SUBSHELL)
1
kashif@Zen:~$ (echo $BASH_SUBSHELL; (echo $BASH_SUBSHELL))
1
2
```

File descriptors

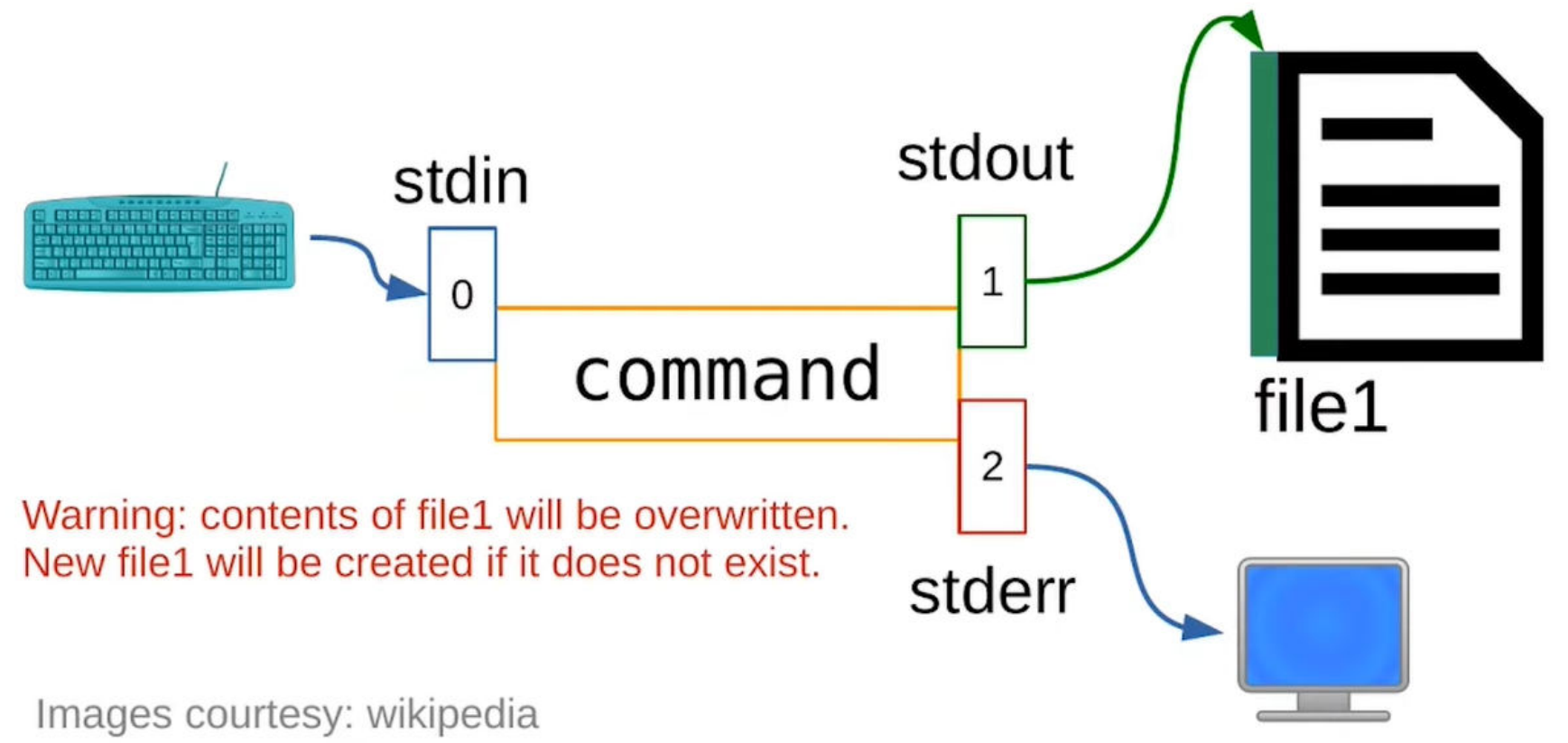


Every command in Linux has 3 file descriptors

- `stdin` (0)
  - It is a pointer to a stream that is coming from the keyboard (or the user input)
- `stdout` (1)
  - Points to the screen where the output is made
- `stderr` (2)
  - Points to the screen where the output is made

`command > file1`

- The output of the `command` should be written to `file1`



Create a file using `cat` command

`cat > filename`

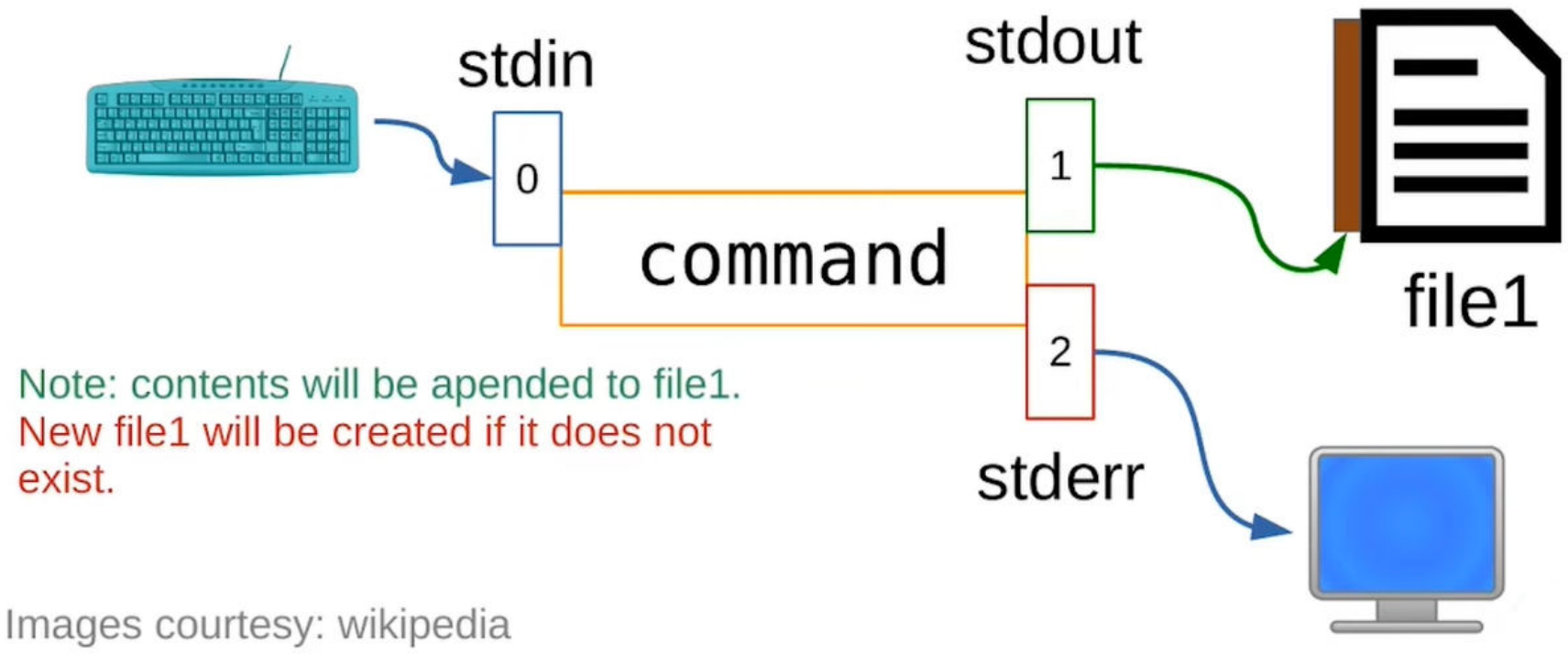
When we type this command, the `cat` command is supposed to receive the input from a file that is listed in the command line, but instead, we left that intentionally blank

So, the `cat` command, instead, reads the content from the `stdin`, i.e. the keyboard

To exit, press `Ctrl + D`

`command >> file1`

- The output of `command` will be appended to `file1`



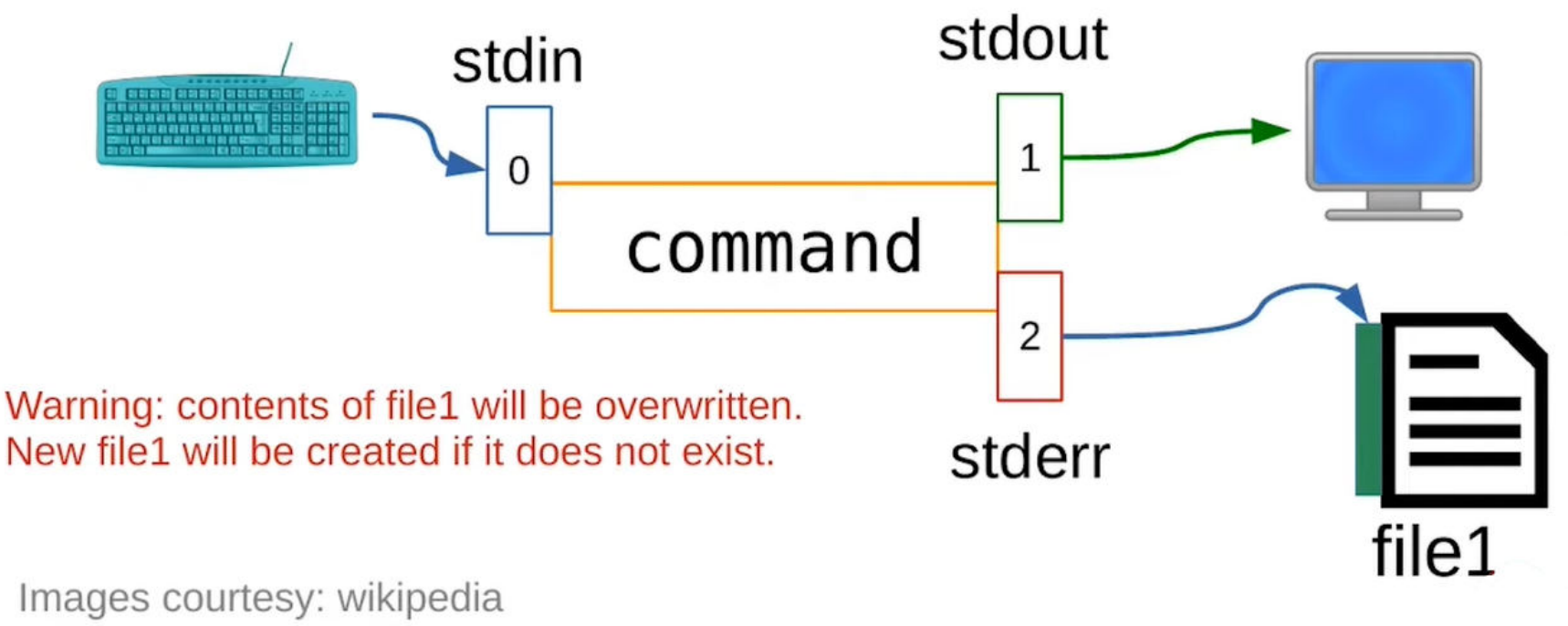
Similarly, we can use `>>` instead of `>` while creating a new file using the `cat` command

Redirections

Type	📒 Lecture
Date	@January 9, 2022
Lecture #	2
Lecture URL	https://youtu.be/BBh69kH_G_Y
Notion URL	https://21f1003586.notion.site/Redirections-734673f36f21448f99de25ccb092c8d4
Week #	3

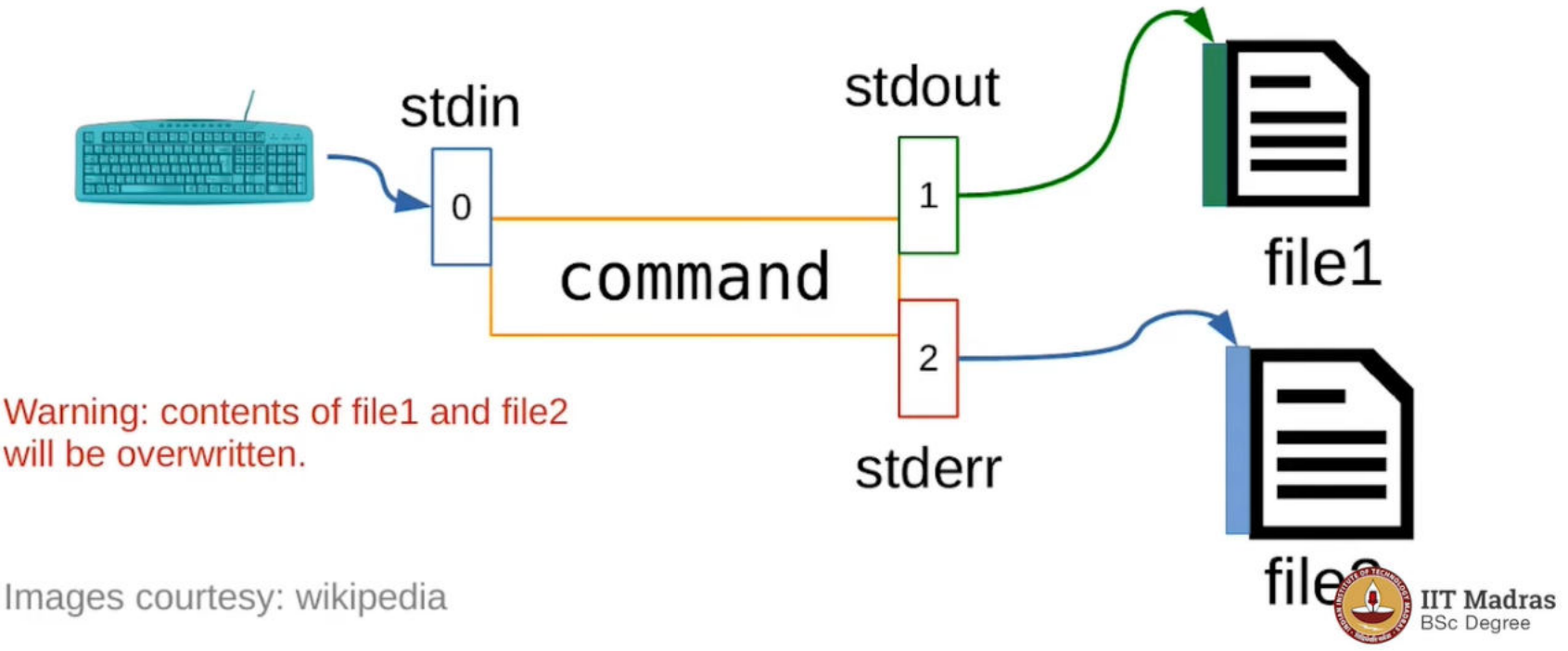
`command 2> file1`

- Redirect the output of the `command` to `stdout`, which is the display in this case
- Redirect the error of the `command` to `file1`



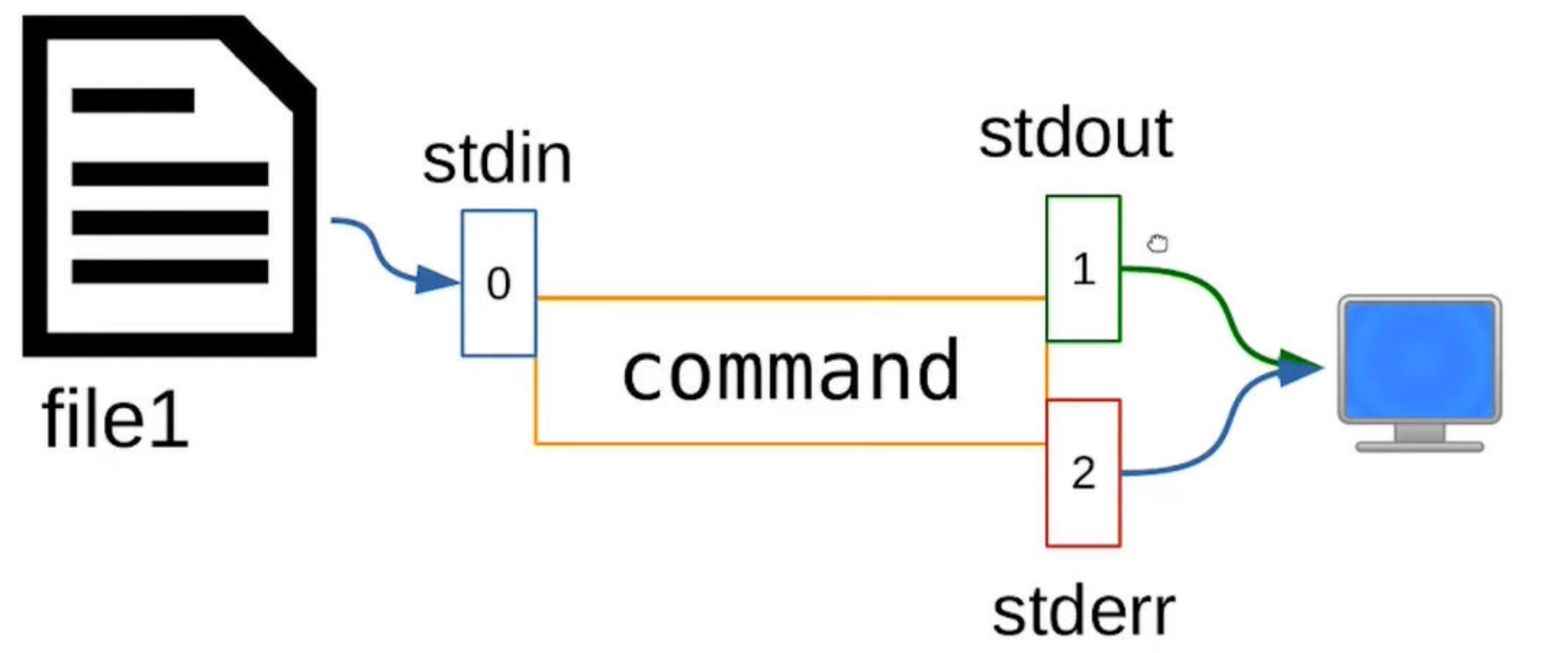
`command > file1 2> file2`

- Redirect the output of the `command` to the `stdout`, i.e. `file1`
- Redirect the error of the `command` to `stderr`, i.e. `file2`



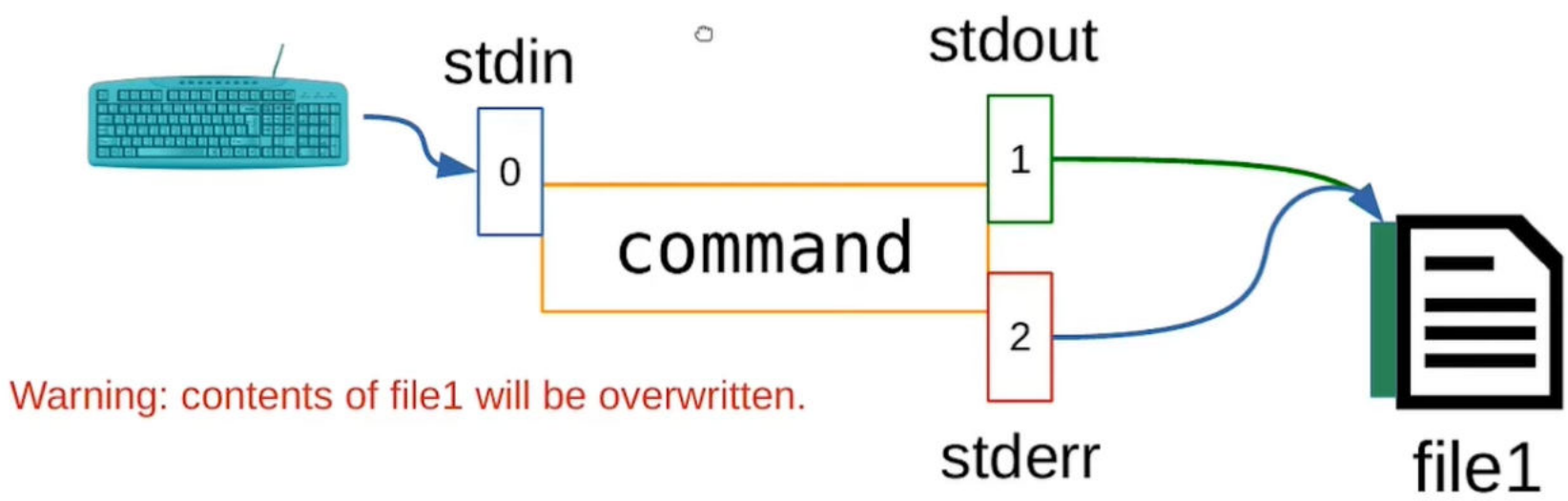
`command < file1`

- Any `command` which takes input from the keyboard, now takes input from `file1`



`command > file1 2>&1`

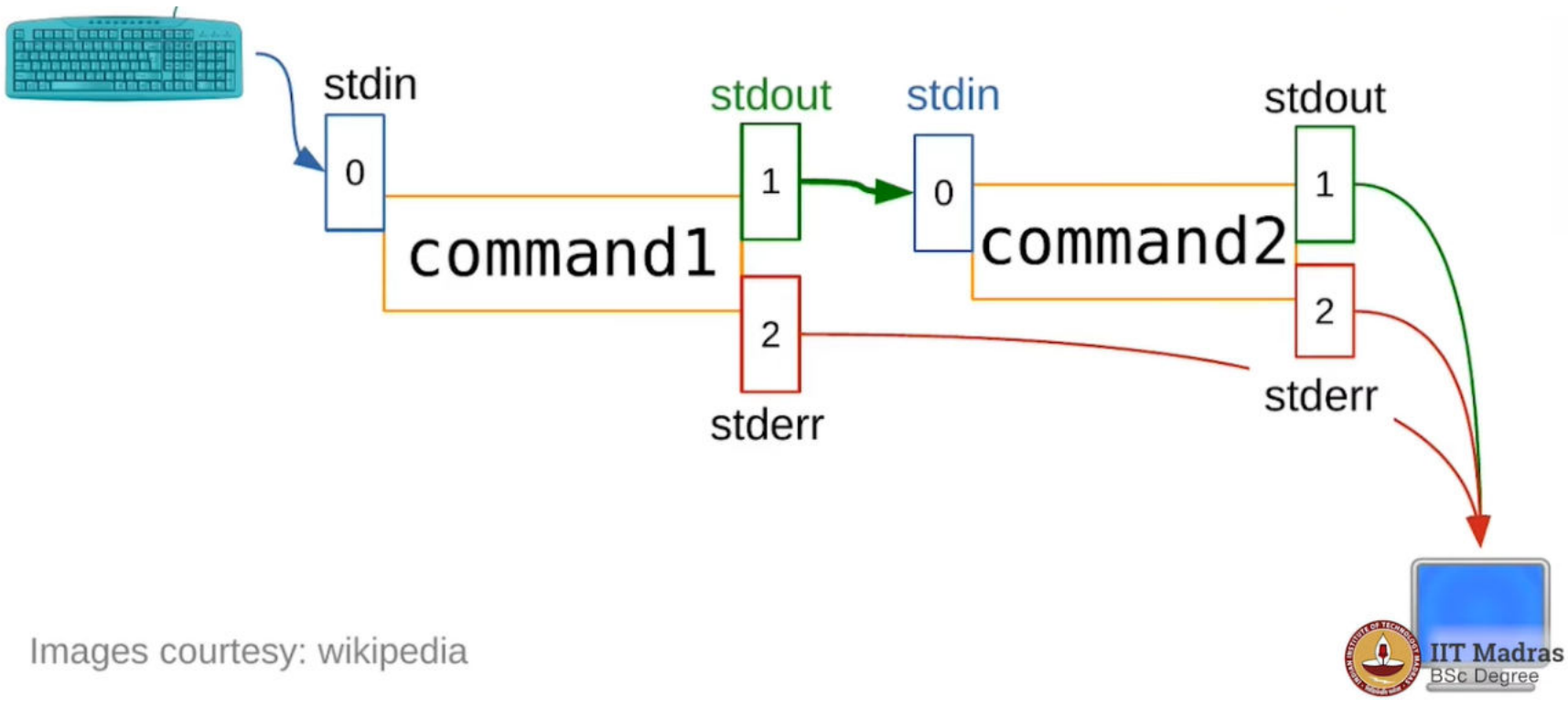
- The output of `command` is written to `file1`
- The error is redirected to stream 1, which is `stdout`



`command1 | command2` → **pipe operator** `|`

- The output of `command1` is sent to `command2` as input

- By default, the `stderr` will output to the display



Count the number of files in the directory `/usr/bin`

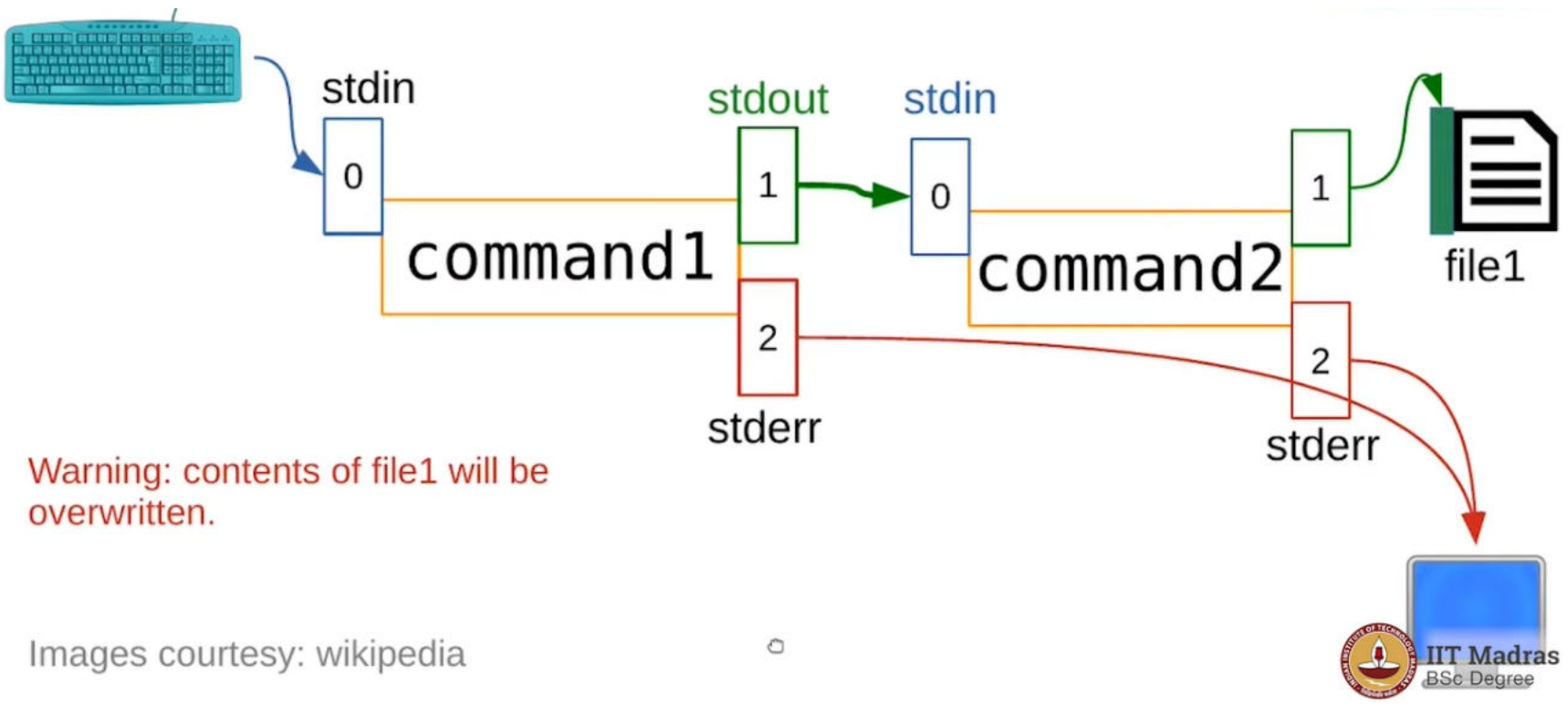
```
kashif@Zen:~$ ls /usr/bin/ | wc -l
1603
```

List the files of `/usr/bin` directory, but use the `less` command to scroll at ease

`ls /usr/bin | less`

`command1 | command2 > file1`

- The `stdout` of `command1` is mapped to `stdin` of `command2`
- The `stdout` of `command2` is written to `file1`
- The `stderr` is output to the display



`/dev/null`

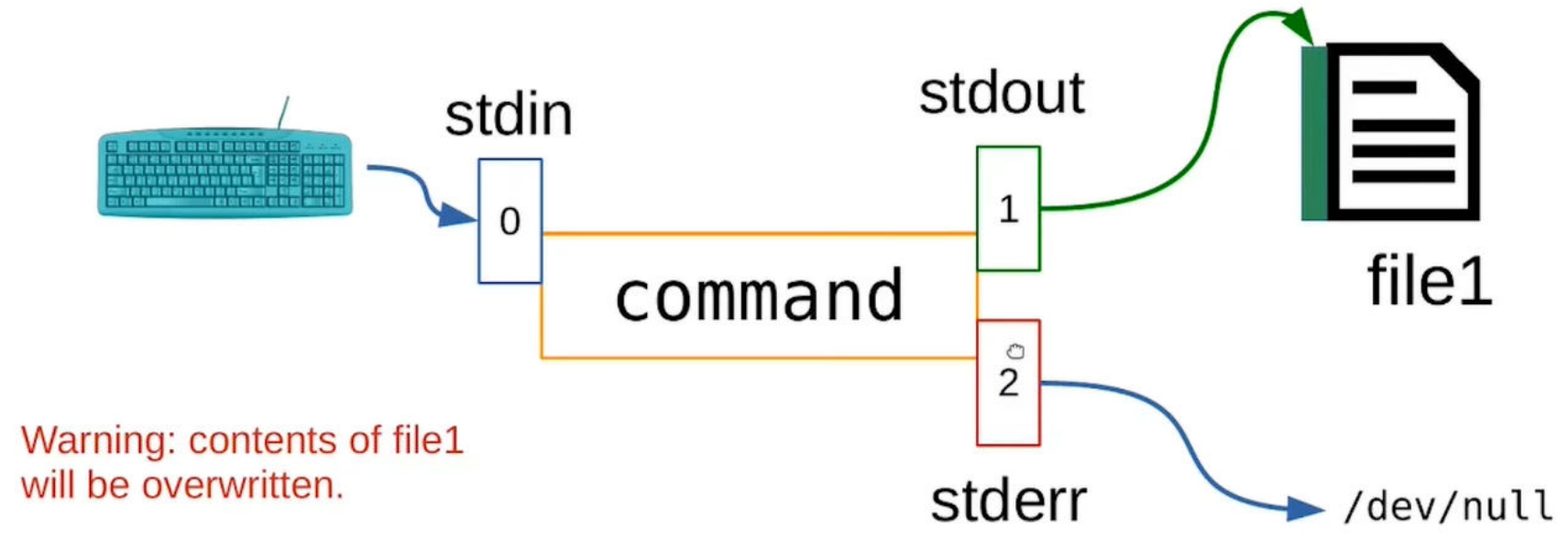
- A sink for output to be discarded
- Use → silent and clean scripts

So, a typical usage looks like ...

`command > file1 2> /dev/null`

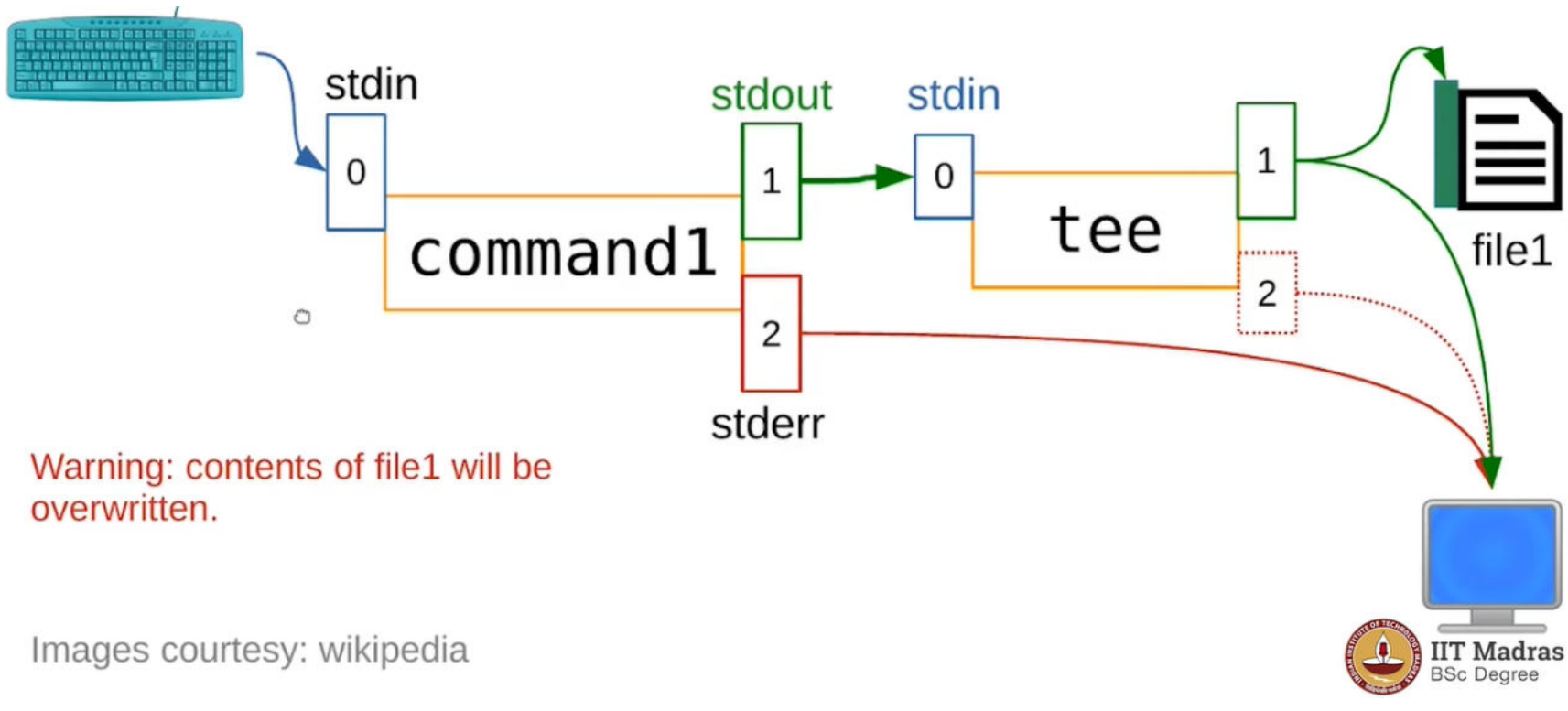
- The output of `command` is written to `file1`

- The `stderr` is written to `/dev/null`, ~~*which gets warped to another dimension*~~



`command1 | tee file1`

- The `tee` command splits the output into 2 streams, one stream is written to the `file1` another, one to the display
  - This command can write to multiple files as well



`diff` command

- This command compares files line-by-line

Usage

```
diff file1 file2
```