Command Line Environment

In shell, we can stop running programs by using keyboard shortcut *CTRL+C*. This will send a *SIGINT* signal to the shell. More explanations on the signals can be found by using `man signal` command on the shell. We can also use *CTRL+Z* shortcut to suspend a program. When we do it, we can actually continue that program whenever we want. For that, we use `jobs` command.

Let's start a program: write *sleep 100* to the shell and press enter. At that point, you are not going to be able to do anything else because the process is running on the *foreground*. If you press *CTRL+Z* and then write *jobs*, you will see that *sleep 100* is *stopped*. Also, at the left side you will see an id, *[1]*. When you write *fg %1*, that program will carry on running on the foreground. If you write *bg %1* that command will carry on running on the background. If you want to really stop it, you can write *kill %1*.

However, there are some signals which can be caught by the program. This is done intentionally. For example when you start python shell on bash, you cannot quit the shell by using *CTRL+C*. It will tell you *KeyboardInterrupt*. This can happen because you can catch *SIGINT* in code. However, some commands cannot be caught. When they are sent, process is killed. Catching is useful because maybe you will want to save the state of the program to somewhere before you quit. You can count the number of SIGINTS and maybe if the user sends it 2 times in a row, you can save and quit.

Running stuff in the background

By using *&* symbol, you can tell bash to run commands in the background. This is useful because as we have seen in the previous example, when a process is running on the foreground, you cannot do anything else. That is why, we sometimes want to run programs in the background. Try it with the same example, *sleep 200 &*.

Terminal Multiplexer

Instead of starting multiple terminals or tabs in a terminal, you can use a single shell screen to use multiple shells. Most popular two programs are *screen* and *tmux*. Tmux is more user friendly and more popular. It is a very functional system.

There are 3 main ideas in tmux. Sessions, windows and panes. You create a session by using `tmux -new -t sess.name`. In there, you can create multiple windows, name them, and then see multiple windows at the same screen. There are shortcuts online and good cheatsheets for it.

In a graphical operating system, one may not see this as useful. However do not forget that if you are going to use a remote server and connect to it via ssh, you are not going to be able to open multiple tabs in a terminal because you will be in command line environment. Tmux is a good application to know in those cases. One, you can run multiple programs in multiple tmux windows. You will be sure that your command is not going to be killed when you disconnect from the server. You can create several sessions for different projects and make sure that you do not mistake them.

There is a command line control in tmux but keyboard shortcuts are much faster.

Dotfiles

These are files that helps you customize some applications in the shell. For example for bash, we have *.bashrc* and for zsh, we have *.zshrc*. These all start with a dot and they are hidden from the simple *ls* command. To see them, you need to run *ls -a*. Dotfiles are also used for some programs, not only for shell customization. For example you can use one for vim, tmux or git. You can configure these programs to your own needs. There are a lot of examples of dotfiles to check from. These files are generally found in ~ (home) directory.

Keep in mind that instead of changing the original files, you can create a dotfiles directory for yourself and use *symlinks* to link the original dotfiles to your own without changing them. Creating a good system for your dotfiles is important if you heavily customize your shells or applications.