Command-line Text Editors

We talked about three most popular command-line text editors: **nano**, **vi** (vim) and **emacs**.
So let's first look at **nano**. **Nano** is a simple text-editor that's friendly to newcomers. You can pretty much figure out how to use it in few seconds. To open a file in nano, write nano followed by the filename.

```
$ nano filename
```

Now you can just start typing. When you are done, press **Ctrl+x** to exit the file. If you didn't save it, you will be prompted to do so. Than you can just type **y** for yes and press enter to confirm the filename.
If you just want to save, without exiting, press **Ctrl+o** to "write out" the file.
You can get help on all of these shortcuts by pressing **Ctrl+g**.
To cut out a line of text, use **Ctrl+k**. Then paste it with **Ctrl+U**.
Go to the specific line number with **Ctrl+t** which will open a prompt for you to type in the wanted line number.
You can search and replace text with **Ctrl+r**.

Here is the nano manual.

The **vi** editor is a little bit more complicated.

```
$ vi filename
```

I usually use **vim** in the command-line, which is the improved version of **vi**. This editor functions in modes. *Command mode* enables you to execute command and administrate files. This would include all of the file editing features you would usually use in any other graphical editor. *Insert mode* is just for inserting text into the file. This mode would be something like just opening up nano.
To close a file in **vi** type the **:q!** in the command mode. If you want to save it

also, then type **:wq** instead. **:w** is just for saving a file. Press **i** to enter the insert mode in which you can type in the file content. To exit the mode press **Esc**.
Now this is just for entering, editing and exiting a file. The power of *vi* and *vim* lies in its shortcuts. You have shortcuts for virtually anything you can imagine. These shortcuts can be used make your workflow more faster and productive. It is said that in vi, you can edit text with the speed you think of it. No more useless mouse movements to slow you down. There is also a visual and visual block mode which give you more functionality. This is a really broad topic and I urge you to get into it, because I am a satisfied **vim** user myself. There are a lot of books written on *vi* and *vim*.
But don't just read. Start using it every day and slowly incorporate more and more shortcuts and features, as the time goes by. It will seem hard at the beginning, but you will be better off in the long run.
Here are some shortcuts you can use when you enter a file with **vim**.
You need to be in the *command mode*.

Purpose	Keys
j	go down to the next line
h	go one character to the left
l	go one character to the right
k	go up to the previous line

These letters are used like arrow keys for moving.
However, if you want to move more than one line or character, append a number before typing the letter. For example **5j** will get you down 5 lines.

Purpose	Keys
w	go to the next word
b	go to the previous word
$	go to the end of the line
0	go to the beginning of the line

These can also be prepended with numbers, e.g. **4w** will skip 4 words in the line.

Vim is really extendable and you can enable a lot of built-in options. For example, you can first set the line numbers with **:set number** and press **Return**. You can also enable **:set relativenumber** to set relative numbers.
When you do this, only the current line you are on will show the exact line number. All of the other line numbers will show how many lines are between them and the current line.
Try it! Here is a file as an example.

```
Hello there
this is
an example
file
```

When I turn on the **number** option, we will get the line numbers.

```
1 Hello there
2 this is
3 an example
4 file
```

Now I will also turn on the **relativenumber** option.

```
  2 Hello there
  1 this is
  3 an example
  1 file
```

You can see that a cursor is placed on the third line of the file. That is the only line that really shows the file line number. All of the other lines are showing the line distance. This is useful, because if you want to quickly skip to a far away line in a long file, you can see how much lines you have to skip. If you need to go down 10 lines, you can just type **10j** and you are there.

Purpose	Keys
u	undo
Ctrl + r	redo
dd	delete the current line
d$	delete the line from the current position to the end of the line
d0	delete the line from the current position to the beginning of the line

As you can see, the key bindings make sense. We saw that you can move to the end of the line with **$**, so to delete to the end of the line, we use **d$**.
This is just the navigation. When you are ready to edit, type **i** to enter the insert mode at the cursor position. **O** will start the insert mode on the next line.

Lastly, we mentioned **emacs**. If you are hard core programmer and you are not using **vim**, you are probably using **emacs**. GNU Emacs is extensible using a Turing complete programming language. It is not just a text editor, you can pretty much to anything with it.

In the video I showed you the basics of entering **emacs**, editing and exiting the file. You need to install it, because it will not come preinstalled on many operating systems.
The **buffer** in **emacs** is the area in which you write in text. The **buffer** is contained inside of a **frame** (which would people usually call a window).
Emacs also has editing modes. **Buffer** will have one **major mode** and any number of **minor modes**. Modes are the way of switching between various key bindings and features.
When you are done typing, you can press **Ctrl + x Ctrl + s** to save a file.
When you start looking up all of the *key bindings*, note that all of them will be prepended with either **C** or **M**. **C** is the **Ctrl** key and **M** is **meta** key (usually mapped to **Alt**).
This editor also has a steep learning curve and you always need to keep learning and extending to new options and features.