

GNU EMACS

Editor MACroS

Editors

Linux has many (hundreds) of options for editing text on the command line.

They range from single line editors (ed) to colossal “operating systems” like Emacs.

“Emacs was never such a great text editor, but its a fantastic operating system.” - Unknown

Nano (pico) first

EMACS is extraordinarily powerful and useful.
Some prefer the simple life.

Enter stage left: nano (or pico on eniac). Very few commands. You can type and move around in the file and use some basic save, open, find/replace hotkeys. That's it.

Nano (Pico)'s commands

Control-o -> save

Control-w -> find (control-r -> replace [Nano only])

Control-x -> quit

Control-c -> show line number

Emacs Buffers

When Emacs opens a file it copies it into a buffer inside Emacs. When you edit inside emacs you aren't touching the file but merely the buffer.

Emacs will auto-save the buffer now and then. You can also use the save command to flush the buffer and overwrite the original file.

Can also drop the buffer without touching the original file!

Emacs can edit many buffers at once and do fun things with that idea.

Using keystrokes as your guide

One of the hardest adjustments to a console based text editor like emacs will be learning hundreds of new commands and their shortcuts.

Particularly important for emacs – its the king of shortcut hell.

Ex: saving a file: Control-x, Control-s

WHAT? A DOUBLE hotkey?

Essential commands

Control-x Control-s -> save

Control-x Control-f -> open

Control-x Control-c -> quit

Control-_ (aka Control-underscore) -> Undo

Control-g -> cancel current action

Setting a mark

Emacs highly discourages the use of the mouse.

You can select text using just the keyboard!

Control space to begin selecting text (set a mark)

Move using arrow keys or other hotkeys to other location in the buffer.

Can then do commands which affect the mark.

e.g. Control-w -> the equivalent of 'cut'.

Control-y -> a pseudo-equivalent of paste

Emacs as an IDE

Emacs was meant for code!

Has excellent auto-indent and parenthesis and auto-complete support.

Has “modes” for different languages to apply language specific settings, syntax highlighting and features.

Emacs shell

For every hotkey in Emacs there is an actual command you can type into the emacs prompt.

The prompt is available by pressing Meta-x. (Meta is the old-fashioned name for Alt.)

The Emacs prompt or shell is very similar to bash in that it has tab completion.

It will also tell you the hotkey for any command you type after you run it.

Emacs integration

Emacs also integrates nicely with other tools (as any good IDE should).

E.g. GDB, Subversion etc.

Vi



vi is another very highly used editor which is extremely different from emacs.

If you find yourself hating Emacs give vi (or vim) a try.

Shell, Pipes, Processes and Redirection

Wait, you can do more with the command line?

* A shell is a command line

A shell is an environment for human command input and processing. (You type into it and it does stuff.)

Examples of commonly used shells:

sh, bash, zsh, ksh, csh

A “terminal” is an application which holds a shell.
xterm, aterm, eterm, gnome-terminal, putty, screen
etc.

Bash

The most commonly used shell (and the default on eniac) is bash. You can type “chsh” (CHange SHell) to change your default shell.

Bash does more than just execute your commands.

Type `le<TAB>`. Bash is cool huh?

More things bash (and all shells) can do

Wild card resolution.

Looking for just text files?

```
/home/zgold/ # ls *.txt
```

```
test1.txt test2.txt
```

Just music by Switchfoot?

```
/media/music/ # ls Switchfoot*.ogg
```

Wildcards – what actually happens

ls doesn't actually receive the arguments “*.txt”.

Instead, bash automatically expands the expression.

Other tricks:

ranging: ls SomeNumberedStuff[1-23]

list a lot of directories: ls */*

*An aside: heads tails and cats

cat (concatenation) : read from any number of files concatenate them into a single stream. When used from the command line it writes data to the console.

Common usage: to read files.

```
/home/zgold #: cat ShoppingList.txt  
Milk, Cheese, Cereal, Cookies
```

*Heads and tails

head: returns the first X lines from a file or stream.

```
/home/zgold # head LongList.txt
```

beginning

of

tail: returns the last X lines of a file or stream.

```
/home/zgold # tail LongList.txt
```

some

list

Sorting

Sort sorts data

```
/home/zgold # sort LongsList.txt
```

beginning

of

list

some

But what if you want to print the first 5 items from the sorted list of the last 10 items in a file?

Tailsorthead ?

Modularity

It seems, and indeed would be, silly to have lots of commands like “tailsorthead” and “sortheadtail” etc. It also violates principles of modularity.


So how do we get around being able to combine multiple commands without making new ones?

Atomicity

Notice that the operation we want to perform, `tailsorthead` is actually just the three commands in sequence. If we could perform the first on the data, then the second, then the third we have achieved what we want. So why not do just that?

```
/home/zgold # tail -n 50 LongList.txt | sort | head -n  
5
```

Pipes



So what did we just do? We “piped” the output of one command into another. We took the output from the first, and made it the input of the second. We took the output from the second and made it the input to the third.

*Redirection

Alright, so we did some cool stuff with the data... but what if we want to save it?

Well, we said the console is just an output stream right? Well so are files.

Shells allow you to redirect inbetween streams. So all we have to do is redirect the console output (hence forth *STDOUT*) to a file with the “>” *operator*.

```
/home/zgold/ # tail LongList.txt | sort | head -n 5 > newFile.txt
```

Processes and Daemons

Every command you run on a command line “forks” into a new process which is running on the computer.

Most processes are a result of the user starting some form of program, however sometimes, for example at system bootup, processes are started that are meant to be running in the background.

These processes usually provide some form of data service or maintenance task.

Such long lifetime, non-GUI based processes are typically referred to as daemons (or services on Windows).

Every Linux distribution has some mechanism for managing running daemons.

Command Line handing of processes

Usually your shell waits for that process to exit before giving you a new prompt.

However you can fork it to the “background” by using the “&” command and continue working immediately.

```
/home/zgold # somethingLong &
```

```
[1] 12345 <-- this is the process ID of the backgrounded process
```

```
/home/zgold #
```

More on processes

ps : List current processes (arguments 'ax' and 'aux' very common)

kill #### - kill a process by process id

fg – foreground a backgrounded / paused process

bg – send to the background a paused process

Don't forget those man pages!

Appendix

Wildcard

Head

Tail

Sort

Pipe

Redirect

Process forking

Daemon