Setting up a Linux Development Environment

Jesus Ramos

Panther Linux Users Group

Getting the right tools for the job

Text Editor

You have to write your code somewhere right? These are your main weapon for getting work done in a *nix environment.

The Shell and Other Tools

This is your direct access to all the tools you need to get stuff done from compilers to interpreters to random utilities that make your life easier.

No IDE?!?!

IDE's while convenient are not the best tools for development.

No IDE?!?!

IDE's while convenient are not the best tools for development.

Portability Problems

It's hard to take code written in one IDE and move it to another, it's equally as hard to make it work across different operating systems. Also because of IDE specific project management structures, managing a project can become a nightmare without it.

No IDE?!?!

IDE's while convenient are not the best tools for development.

Portability Problems

It's hard to take code written in one IDE and move it to another, it's equally as hard to make it work across different operating systems. Also because of IDE specific project management structures, managing a project can become a nightmare without it.

IDE's are heavy

What I mean by this is that IDE's while having some useful features are generally very large in size and require quite a bit of processing power and memory to run (ex. Visual Studio).

No IDE?!?! (cont.)

No IDE?!?! (cont.)

Intellisense

While generally a useful coding feature, it's actually detrimental to coding if you become too reliant on it. Why remember anything if all you have to do is press TAB and have the IDE complete it for you or just scroll up and down a menu until you find something that looks right.

No IDE?!?! (cont.)

Intellisense

While generally a useful coding feature, it's actually detrimental to coding if you become too reliant on it. Why remember anything if all you have to do is press TAB and have the IDE complete it for you or just scroll up and down a menu until you find something that looks right.

Code Navigation

Code navigation in an IDE tends not to be very good. Symbol look ups are slow when navigating methods/functions. Jumping around in the code uses clunky hotkeys and code formatting generally differs between IDE's and can be a pain to setup.

Choose your weapon

Popular text editors.

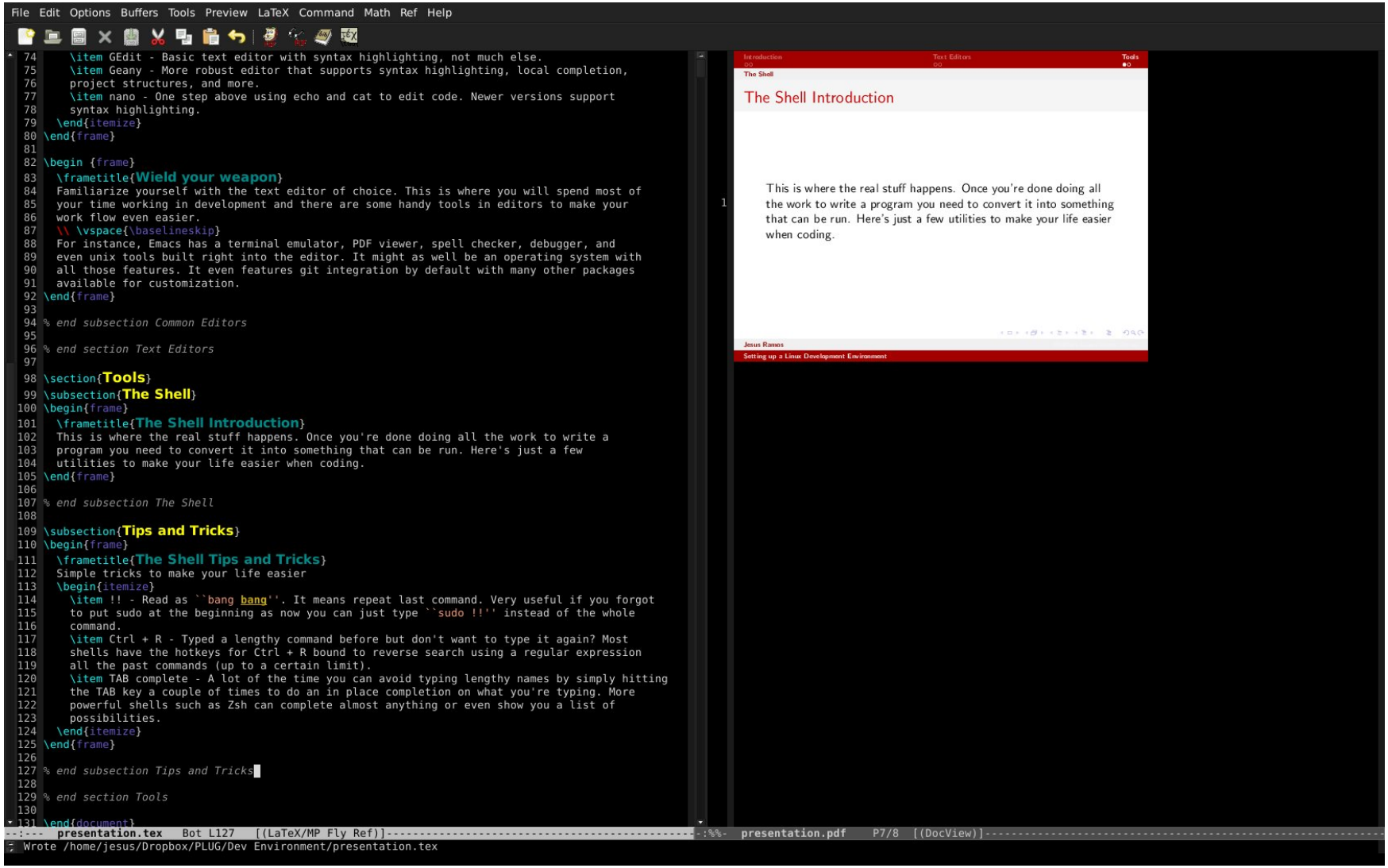
- Emacs - Hotkeys galore, based on your hands never leaving the main part of the keyboard while you type. Can also double as an operating system.
- ViM - Has 2 modes, one for editing and another for navigating code. Replaced Emacs as the default text editor for most *nix systems due to its size.
- GEdit - Basic text editor with syntax highlighting, not much else.
- Geany - More robust editor that supports syntax highlighting, local completion, project structures, and more.
- nano - One step above using echo and cat to edit code. Newer versions support syntax highlighting.

Wield your weapon

Familiarize yourself with the text editor of choice. This is where you will spend most of your time working in development and there are some handy tools in editors to make your work flow even easier.

For instance, Emacs has a terminal emulator, PDF viewer, spell checker, debugger, and even Unix tools built right into the editor. It might as well be an operating system with all those features. It even features git integration by default with many other packages available for customization. It's all useless if you don't use it or know how to use it.

Emacs in action



The Shell Introduction

This is where the real stuff happens. Once you're done doing all the work to write a program you need to convert it into something that can be run. Here's just a few tricks to make your life easier when coding.

The Shell Tips and Tricks

Simple tricks to make your life easier

- !! - Read as "bang bang". It means repeat last command. Very useful if you forgot to put sudo at the beginning as now you can just type `sudo !!` instead of the whole command.
- Ctrl + R - Typed a lengthy command before but don't want to type it again? Most shells have the hotkeys for Ctrl + R bound to reverse search using a regular expression all the past commands (up to a certain limit).
- TAB complete - A lot of the time you can avoid typing lengthy names by simply hitting the TAB key a couple of times to do an in place completion on what you're typing. More powerful shells such as Zsh can complete almost anything or even show you a list of possibilities.

Building your code easily

Most if not all unix systems come by default with the make utility.

This is a more standard approach to having “projects” as it is more widely supported than IDE specific project structures.

The make tool is also very powerful and allows you to make your project cross compatible very easily.

Debugging Code

Simply throwing random print statements in your code to debug is not always the best approach, especially if you're getting segmentation faults or even memory issues.

Luckily for us there are plenty some nice tools to help us find those errors.

GDB

GDB ("GNU Project Debugger") is the main utility for debugging just about any code.

It's a command line utility so it tends to have a pretty big learning curve but there are some good front ends for it such as the old but reliable Data Display Debugger (DDD) and Emacs even has a built in GDB mode for debugging that gives you an interactive console as well as the ability to set breakpoints directly in the code you're viewing.

GDB (cont.)

GDB supports all the standard features of debuggers such as watching variables, setting breakpoints, and even inspecting random memory locations. Newer versions even support debugging of Java code running in the JVM.

Valgrind

Although less often used than GDB, Valgrind is an essential tool for development in languages that support direct memory access because of the likelihood of a memory error.

Valgrind can be set up to find memory leaks, track out of bounds access on allocated memory, and even make sure that your pointer accesses are "safe". Combine this with GDB and you're bound to find the problem.

Version Control Intro

Version control allows you to keep track of incremental changes to your code/document or really anything you're working on and allow you at any time to see changes you've made and possibly change things back to previous states in case something goes wrong.

It's also a good way to have backups of your code in case something goes wrong and a way of measuring progress on something you might be working on.

This also makes working in teams easier as you can now work on separate copies and merge everything together when you're done in a more streamlined fashion.

Git and Subversion

The two most popular version control systems out there are Git and Subversion.

Git is very easy to setup as it only requires installing the git-core packages from your distribution's repository and can work locally without needing a remote server to house all the files.

Github is a site that offers free hosting for Git repositories and can be a great place to store projects that you may be working on, although free accounts can only have public repositories.

Additional Recommended Packages

- Color-theme for Emacs – A useful package for changing the default color scheme of the editor (save battery life or make it easier on your eyes).
- LaTeX – Markup language for presentations/documents which is very easy to use and sometimes more convenient than Word or related programs.
- Auctex – Editing major mode in Emacs for LaTeX for more advanced highlighting and fontifying.
- Zsh – Very powerful shell with inline regular expression expansion for making it easier to navigate your Linux environment.

Pretty easy right?

Linux development is actually pretty simple, you really only need a text editor and a compiler but there are plenty of other tools out there to make your life a lot easier.

For additional questions you can drop by the PLUG IRC chat at http://plug.cs.fiu.edu/chat/ or you can contact me at jramo028@fiu.edu