Arch Linux Tutorial

Ginger Gengar

12th July 2022

Contents

0.1 Rationale

Below explains why this tutorial exists. This tutorial is only valid for Arch Linux that was personally configured.

0.1.1 Why Use a Computer?

I cannot certainly speak for all users, I use a computer to help me do abstract work. Mathematics is the backbone of most major engineering fields. A lot of things in mathematics can be solved using analytical hand-written solutions, but many real-life problems cannot be solved that way. They must be solved numerically, by manual iteration. This is fine if your problem is small, but once your problem gets large enough, it becomes impossible to do by hand and instead, you have to turn to computers.

What kind of mathematical problems am I referring to? I am referring to ordinary differential equations, large matrix equations, tensor operation and even experimental symbollic computation. Why are these problems important? Solving these mathematical problems allows us to construct bridges, skyscrapers, train networks, cars, ships, aircrafts, rocketry, and most modern technologies we have today. Technology's backbone is theoretical engineering and theoretical engineering's backbone is mathematics, and mathematics can then be solved using general-purpose computers. So computers in this regard represent the last step of computing answers to. Computers are NOT A SUBSTITUTE for HUMAN THOUGHT.

What other things can be done by tinkering with a computer? Automation of manufacturing, moving actuators, flying aircrafts and driving cars. Lower level electronics that can marginally be called computers also perform alot of vital tasks in multiple industries, though I am not an expert in this regard, I deal more with the theoretical computations so far. There are many other applications of computer beyond what I have discussed, but the above are already very good reasons to learn computing.

0.1.2 Why Use Linux?

One of the most astounding things I still remember to this day switching from Windows to Linux (Ubuntu) was the existence of package managers. For some reason my machine did not come with mingw, which made it incapable of compiling `C`. I spent almost 2 days trying to install mingw navigating through old sites, and being concerned of accidentally installing malware. Eventually I failed to install mingw. In Linux (Ubuntu) though, the `C` compiler `gcc` was pre-installed, and even if it wasn't, a simple command to the package manager solves all of 2 days worth of problems in just 2 minutes, not even.

I also attempted to install a linear algebra library called `armadillo`. I also wanted to try out another linear algebra library `eigen`. I had to go through so many hoops installing all of the dependencies manually for `armadillo` such as `BLAS`, `SuperLU`, and others. With a linux system, a simple package manager command installs everything perfectly. So the point is, the package manager in Linux is absolutely necessary for any real programming work and is one of many great reasons to use Linux.

It didn't mean much when I first started Linux, but Linux is largely customizable. The text editor, kernel version, window manager, compositor, shell, and many other utilities can be configured and customized to the user's liking. This makes the Linux operating system an incredibly personal and efficient machine. Think about the Linux operating system as a lego piece whose parts can be defined by the user themselves and chosen. This also means that no 2 Linux operating system are exactly alike because each user has their own configurations and choice of programs.

The customizability of Linux also permits a variety of workflows, including `cli`-based workflows. `cli` stands for command line interface. This is a way to use the computer that abandons the mouse altogether and does everything from just the keyboard. It is faster and easier to do many tasks that way. In this tutorial, `cli` would be used extensively. The last reason I have is an ideological one, which is that the Linux operating system is part of a free operating system. Free here as define by the `GNU` does not necessarily mean free of cost (although Linux is free of cost) but instead, free here means the users are free to examine the system in great detail, learn and expand it on their own. This is important because it gives the users the reigns to how computing is done. Users can verify no malware is pre-installed on the machine, and that the computer performs up to user expectations. Users are also not dependent on corporates which is good.

0.1.3 Why Arch Linux?

A Linux distribution just means a version of Linux. Arch Linux is a Linux distribution, which means that Arch Linux is a version of the Linux operating system. To be clear, all Arch Linux systems are part of the Linux family, but not all Linux systems is Arch Linux. There are a few reasons why Arch Linux is a good Linux operating system:

Arch Linux supports both binary installs and building from source. Compiled programs are translated from ther text form into machine code, which are 1's and 0's in human-unreadable form. The act of translating text format into machine code is called compilation. Binary installation is downloading those raw machine code and putting them in the appropriate directory inside the system. Building from source or source installation is downloading the human-readable text form, and then performing the compilation process on the user's machine before installing the resulting binary in the user's machine. Building from source always takes longer than binary install, but has the potential of resulting in a higher performing installed program. Arch Linux supports both types of installs, which makes it very flexible and can install a very large variety of programs.

Arch Linux is also very minimal. Minimal means that the base Arch Linux system has very few things installed in it. This means that users are free to build the system from the ground up, specifying the browser, terminal, window manager, compositor, status bar, text editor and other details themselves. A minimal system IS NOT IDEAL for users who believe an operating system should work out of the box. In a minimal system, one has to specify everything themselves, which is where the customization power of Linux is best revealed. Users who want to trade customization power for convenience should just try another distribution like Ubuntu, Linux Mint, Manjaro, etc. A non-minimal system has drawbacks when the user attempts to customize it because then the user has to uninstall unnecessary pre-installed packages, and then figure out the relationship between different packages and how to break their link and so on, which is more complicated than just configuring the system on your own. By configuring the system on your own, at least the user knows the different relationship bewteen different softwares they themselves have installed.

0.2 Program Configuration

In this section, we will learn how to configure basic programs.

0.2.1 Basic Terminal Usage

Starting programs, writing, and almost anything can be done through the terminal. To open a new terminal press:

```
super + enter
```

The super key is the command key, or the windows key on a typical keyboard. A new terminal instance should appear on pressing the keybinding above. To quit the terminal instances, press:

```
ctrl + shift + w
```

These keybindings have not been changed from the default in kitty. Kitty has tabs and windows. Windows are inside a tab. To open a new tab, press:

```
ctrl + shift + t
```

To navigate to the tab on the left

```
ctrl + shift + left
```

To navigate to the tab on the right:

```
ctrl + shift + right
```

Within a single tab, it is possible to open multiple windows. To do this, press:

```
ctrl + shift + enter
```

To navigate to the left window,

```
ctrl + shift + [
```

To navigate to the right window,

```
ctrl + shift + ]
```

The keybinding `ctrl + shift + w` can close windows one at a time and tabs as well, try experimenting with it. To open a new tab between existing tabs, press:

```
ctrl + t
```

This should cover the basics of terminal navigation.

0.2.2 Basic Window Manager Usage

Without anything else open, user resides in the first workspace. To go to the second workspace, press:

```
super + 2
```

To go the third workspace,

```
super + 3
```

Entering a different number will go to that workspace. To go back to the original workspace, press:

```
super + 1
```

Try opening 2 terminals by pressing `super + enter` twice for the two terminals. Try opening multiple tabs on one of them by pressing `ctrl + shift + t` successively for one of the terminals. To navigate between the terminal instances press:

```
super + left
```

Alternatively, one can press:

```
super + right
```

To make the terminal instances tile differently, One can navigate to either one of the terminal instances and press:

```
super + shift + left
```

One can experiment with the following commands for different ways to tile the terminals:

```
super + shift + up
super + shift + down
super + shift + right
```

One can think of the `super + shift` as the keybinding to ”hold and drag” a particular terminal instance. The same logic can be applied to put terminal instances and other programs in different windows. Try navigating to one of the terminal instances and then pressing:

```
super + shift + 2
```

Pressing different numbers will move the terminal instances or other programs to different workspaces. For example, to move a terminal instance to the third workspace, navigate to the terminal instance and press:

```
super + shift + 3
```

The window manager also can force quit a window. To kill a particular window, navigate to it and then press:

```
super + shift + q
```

This should cover the basics of how the window manager works.

0.2.3 Basic Core Utilities

To make a new file use type in the following command in a terminal instance:

```
touch Pokemon.txt
```

`touch` is one of the core utilities in the Linux system, and the command above just means create the file named `Pokemon.txt` inside the current directory. Changing `Pokemon.txt` with another name would change the name of the file you are creating. For example, to create another file named `Digimon.dat` one can simply just type inside the terminal:

```
touch Digimon.dat
```

One difference coming from a `Windows` operating system is that file name extensions like `.txt`, `.dat`, `.exe` in windows tell the operating system what to do with those files. In Linux, file name extensions are more like suggestions for the content inside. The operating system will not care what the extension of those files are. So in a Linux system, naming a file `Pokemon.txt` and `Pokemon.dat` will not change how the computer treats the files. Now to see the list of files and folders in the current directory, invoke the `list` command by typing in the terminal:

```
ls
```

If you do that, then you will see the names of the files we have created in the previous prompts. The way to delete files is to invoke the `remove` command. To delete the file `Pokemon.txt` that is in the current directory, type in:

```
rm Pokemon.txt
```

The `rm` command has to be used with great caution. Deleting a file or directory using the `rm` command IS PERMANENT. Now it is time to create directories. To make a new directory named `PokeDex` invoke the command in the terminal:

```
mkdir PokeDex
```

Using the `list` command will give you a list of all the files and directories in a directory. There are a few ways to delete a directory. The `remove directory` command `rmdir` works if the directory is empty and has nothing in it. The `rm` command with a flag works even if the directory is not empty. To delete the directory `PokeDex`, one can invoke the command:

```
rmdir PokeDex
```

Alternatively, one can invoke the `rm` command to also delete the directory by saying:

```
rm -r PokeDex
```

In the above example, `-r` is a flag. We pass the `-r` flag to `rm` which means remove recursively all directories and files inside the directory `PokeDex`. After covering creation and destruction of directories, it is now important to be able to navigate between different directories. Create the directory again by invoking `mkdir PokeDex`. To navigate inside the directory `PokeDex`, say:

```
cd  PokeDex
```

Here the `cd` command stands for change directory. To go back to the directory that contains `PokeDex`, say:

```
cd ..
```

The command above would go back to where you once were. If you are confused about where you are in a linux system, invoke the command `pwd`. Type into the terminal:

```
pwd
```

The command above stands for print working directory. The command above would print where you are inside the linux system. Now you might be wondering how can I rename, move files in Linux? It is possible with the `mv` command. Suppose we are at the home directory and you only have a single file `Pokemon.txt` and you would like to rename this file into `Digimon.txt`, then you can simply invoke the following command if you are inside the same directory as `Pokemon.txt`:

```
mv Pokemon.txt Digimon.txt
```

Remember that file name extensions in linux do not mean anything, so if you wish, there is no need to even keep the `.txt` extension in the name `Digimon.txt` . Supposing you have a directory named `PokeDex`, by creating it using one the commands earlier and you wish to move the file `Digimon.txt` into the directory `PokeDex`. You can do this by invoking the command:

```
mv Digimon.txt PokeDex
```

You can use `cd` and `ls` to verify this has been done correctly. `mv` is a versatile command, it also works for renaming and moving directories into inside other directories. For example you have 3 directories `Games` and `Work` and `General`. You could move `Games` and `Work` to `General` by invoking the command:

```
mv Games Work General/
```

The `cp` command stands for `copy`. This command can be used to copy directories and files. Here is an example:

```
cp something1 AnotherThing2
```

Here `something1` can be a file, a directory, it can be anything, and `AnotherThing2` is the name of the copy. To make your life easier, just press `tab` to autocomplete the commands in the shell. Another useful command is:

```
clear
```

This command clears the terminal of all the previus commands. There is a special directory that is meant to be user-specific. That is the home directory To go to the home directory, one can say,

```
cd ~
```

with the ~ character. Invoking the command,

```
cd
```

also just immediately jumps to the home directory.

0.2.4 Basic Text Editing

0.2.5 Dotfiles & Config Files

The terminal that is configured in this system is `kitty`. `kitty` by default does not look like it does on the machine. The colors have been changed, and the tab bars have also been changed. The font sizes have been changed and so on and so forth. The keybindings for the window manager like `super + shift + 2` to move something to workspace 2 are also modified from their original versions. How can we tell the program user-specific preferences?

The way users can specify their preferences are by writing files inside the `.config` directory in the home directory. All directories that start with a `.` are commonly known as "dotfiles". Directories that start with a `.` are assumed to be system details that the user does not need to know about. So, a simple naive `ls` command will not show all of the "dotfiles". "dotfiles" are also known as "hidden files". Additional flags can be passed to `ls` to allow it to see the "dotfiles". Navigate to the home directory by invoking the command:

```
cd ~
```

Once there, invoke the `list` command with the flags `-a`. This will specify `list` to show all files inside the directory, including the hidden ones:

```
ls -a
```

This may look a bit ugly, so pass the additional parameter `-l` to make it look like a list:

```
ls -l -a
```

This might take a long time to type, so you can make it shorter by instead invoking:

```
ls -la
```

This is a shorter notation for `ls -l -a`. Once you have performed any of the `list` commands above, you can see the "hidden files" such as `.bashrc` and a `.config`. Navigating to the `.config` directory and then invoking the command `ls` would show all of the different programs. Navigating to one of the program directory would reveal another file. For example, navigate to the `kitty` directory. Inside that directory, there is a file `kitty.conf` and `colors.conf`. `kitty.conf` is where the user specific configurations for the `kitty` terminal resides. Changing the contents of the file `kitty.conf` with a text editor from the previous part changes the behaviour of the `kitty` terminal. The same treatment can be applied to different programs to personalize their behaviours.

What to put inside the config file of each program is all program dependent and I would not reiterate the different options here. For more details for each program, visit the individual program pages and read the appropriate documentation regarding customizing the program.

0.3 Reading & Writing: LaTeX

0.4 Basic Programming

0.5 Shell Scripting & Configurations

0.6 Basic Package Management

0.7 Learning Program Usage

0.8 Miscellaneous

0.8.1 Keybinding Summary

0.8.2 Further Reading

General Arch Linux

The Arch Linux Wiki contains a lot of technical information. When trying to debug the system, the Arch Linux Wiki is a great rescource:
`https://wiki.archlinux.org/`

Kitty Terminal

This is the official page for the kitty terminal:
`https://sw.kovidgoyal.net/kitty/`
The Github source code for the kitty terminal:
`https://github.com/kovidgoyal/kitty`

0.8.3 Art Gallery