Raspberry Pi & Linux Guide

I - What is a Raspberry Pi and Linux?

Raspberry PI

What is it?

The Raspberry Pi is a series of small, affordable, single-board computers developed in the United Kingdom by the Raspberry Pi Foundation in collaboration with Broadcom. It's designed to promote teaching basic computer science and to provide an accessible platform for hobbyists, educators, and students to explore computing, learn programming, and work on various digital projects.

Raspberry Pi Projects

Raspberry Pi projects range from simple learning tools to complex systems. Some popular project ideas include:

- Coding and Computing: Using Raspberry Pi to learn programming languages like Python and Scratch.
- Home Automation: Creating smart home devices that can control lighting, temperature, or security systems.
- Retro Gaming: Building a retro gaming console to play classic games.
- Robotics: Designing robots that can perform various tasks or respond to the environment.
- Media Centers: Setting up a media server to stream videos and music throughout your home.

Raspberry Pi Documentation

To read more about the Raspberry Pi, you can visit the official Raspberry Pi website or explore its Wikipedia page for detailed information on its history, models, and specifications. Additionally, there are numerous online communities and forums where Raspberry Pi enthusiasts share their projects and offer support.

Linux

What is Linux?

- Linux is an open-source operating system (OS) that has been around since the mid-1990s. It's part of the Unix-like family of OSes.
- Unlike proprietary systems like Windows or macOS, Linux is freely redistributable, which means anyone can create their own distribution (distro) for any purpose.
- The core of Linux is the Linux kernel, which manages the CPU, memory, and peripheral devices. However, when people refer to "Linux," they often mean the entire OS, including additional software and tools.
- Linux is everywhere: from smartphones and cars to supercomputers and home appliances. It powers most of the Internet, top supercomputers, and stock exchanges.
- Android, one of the most popular platforms globally, is also based on the Linux kernel.

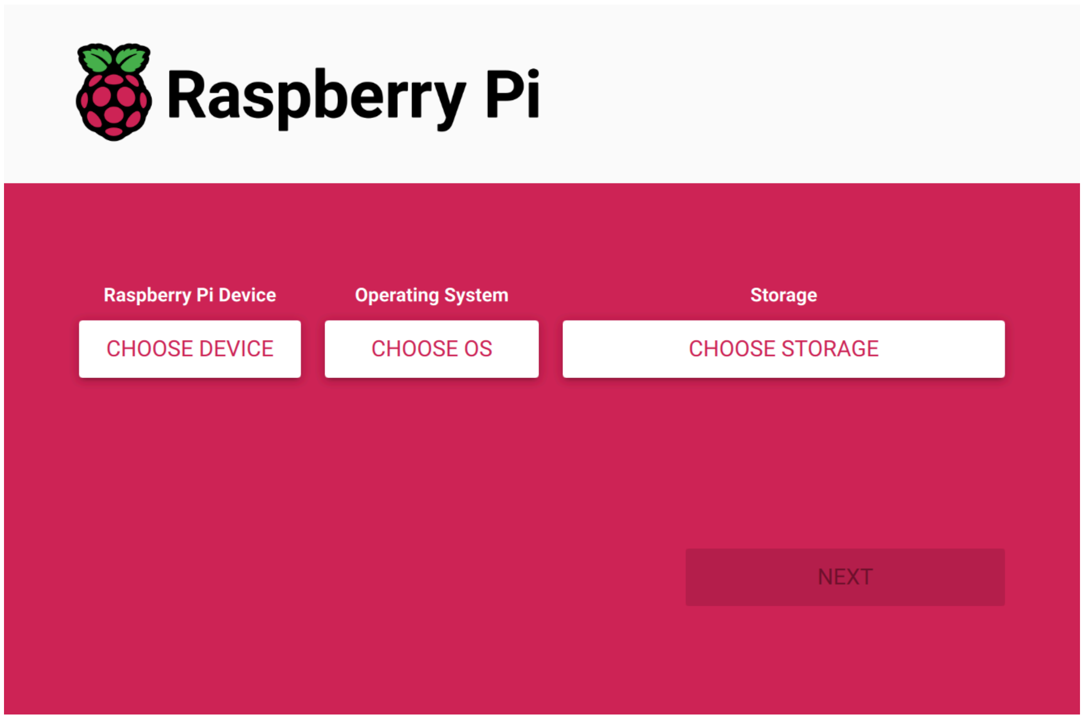
Why does the Pi use Linux?

- The Raspberry Pi Foundation has a unique mission: to help young people learn how to use computers and improve their tech skills.
- They created the Raspberry Pi, an affordable computer (costing less than $50) that can be used in schools. It helps students become more confident with computers and learn programming.
- However, the Raspberry Pi is different from traditional computers, so a specialized OS was needed. That's where Raspberry Pi OS (formerly Raspbian) comes in.
- Raspberry Pi OS is a lightweight Linux distribution designed specifically for Raspberry Pi. It achieves the foundation's goals by providing a compatible and user-friendly environment for learning and development.

II - How to use Linux

Installation

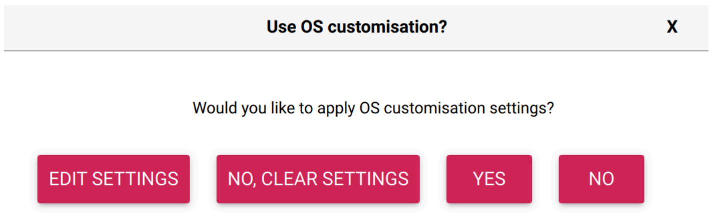
Install the Raspberry Pi Imager on a computer. This is used to install a `.iso` file to a SD card or MicroSD card (depending on your Raspberry Pi model). Launch the application and you'll see the following buttons:



Select your Raspberry Pi device and the storage device connected to your computer.

For the Operating System, if you have a Raspberry Pi 3 or newer, the top option should be OK. Anything older, and you should choose Raspberry Pi OS (Legacy, 32-bit) Lite. Note that this will not install a Desktop Environment (DE), which means everything you'll do is through the terminal. That might sound scary, but don't worry. This guide will go over everything you need to know to hit the ground running.

Once you hit next, you should see the popup below:

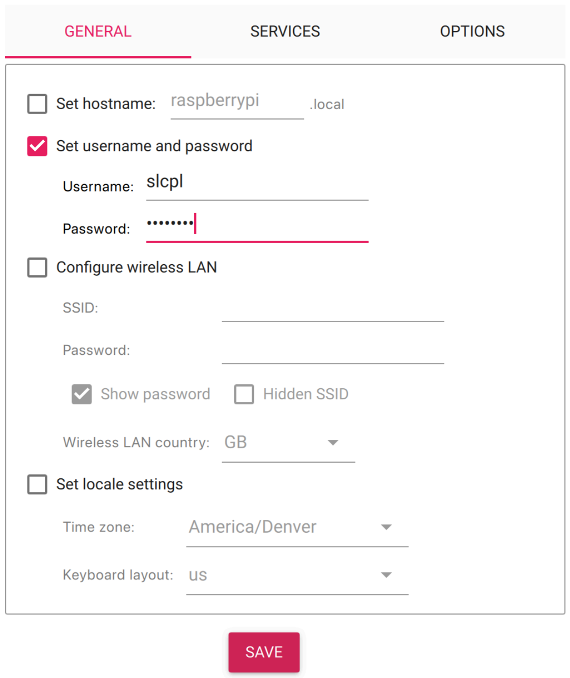


Use OS customisation?

X

Would you like to apply OS customisation settings?

Click **EDIT SETTINGS** and **Set username and password**. This is the only setting you *should* set, but others are available, such as enabling SSH (Secure SHell protocol). For the time being, let's skip those since they're not strictly necessary.


GENERAL SERVICES OPTIONS

[ ] Set hostname: raspberrypi .local

[✓] Set username and password

Username: slcpl

Password: ••••••••

[ ] Configure wireless LAN

SSID:

Password:

[✓] Show password [ ] Hidden SSID

Wireless LAN country: GB

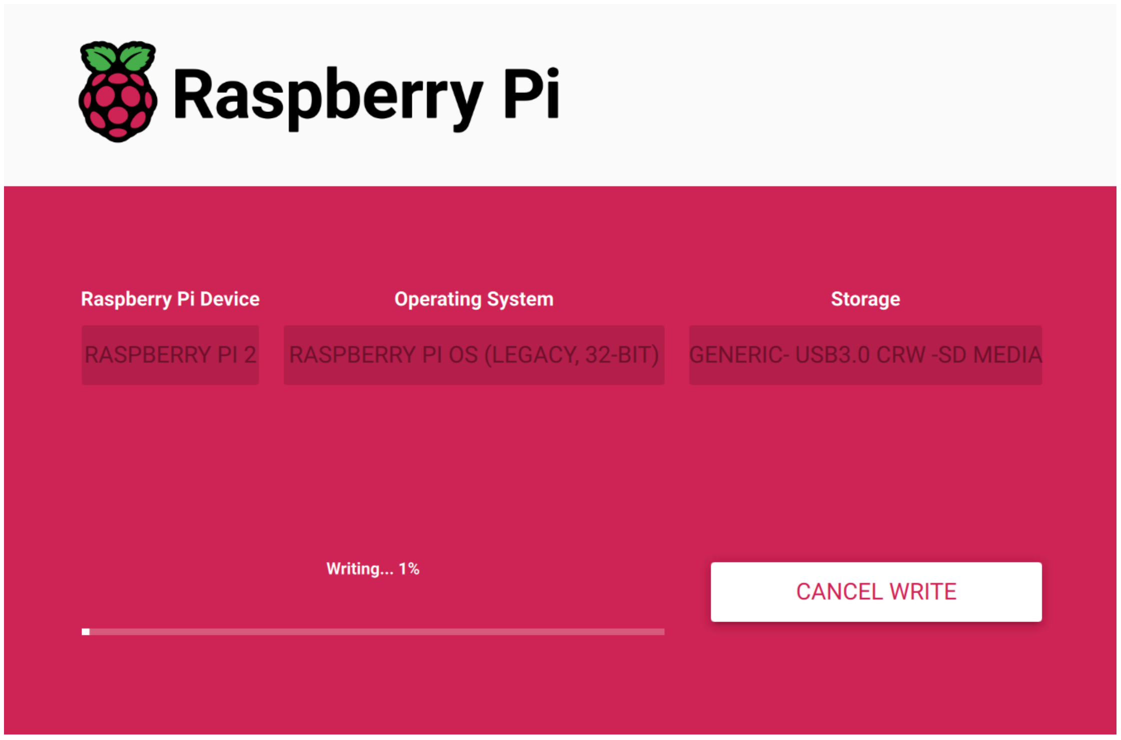
[ ] Set locale settings

Time zone: America/Denver

Keyboard layout: us

SAVE

Now, wait until the installation is complete!


Raspberry Pi

Raspberry Pi Device: RASPBERRY PI 2

Operating System: RASPBERRY PI OS (LEGACY, 32-BIT)

Storage: GENERIC- USB3.0 CRW -SD MEDIA

Writing... 1%

CANCEL WRITE

SSH vs. Local

Using a Raspberry Pi with Linux can be experienced in two distinct ways: through SSH (Secure Shell) or locally by connecting a display directly to the Pi. Here's a comparison of both methods:

- Using SSH:
    - Remote Access: SSH allows you to control your Raspberry Pi from another computer on the same network or even over the internet.
    - Headless Operation: Ideal for running the Raspberry Pi without a monitor, keyboard, or mouse—useful for servers or embedded projects.
    - Convenience: Manage your Pi from the comfort of your main workstation, which is especially handy if the Pi is physically inaccessible.
    - Development: Tools like VS Code can connect to the Pi via SSH, enabling you to write and execute code remotely2.
    - Security: SSH provides encrypted communication, ensuring that your commands and data are secure during transmission1.
- Using Locally:
    - Full Desktop Experience: When a display is connected, you can use the Raspberry Pi's graphical user interface (GUI) for a complete desktop experience.
    - Peripheral Interaction: Directly interact with connected devices such as keyboards, mice, or other USB peripherals.
    - Educational Use: For learning and teaching, having a monitor connected can be more engaging and easier for beginners.
    - No Network Required: Local use doesn't depend on network connectivity, which can be beneficial in environments with poor or no network access.

If you are going to utilize SSH, please be sure to check that option in the settings page when you are installing, though this can be enabled in the future.

III - The Power of the Linux Terminal.

Why use a terminal?

A command is just an application. Some commands open a window (a Graphical User Interface or GUI) while others remain in the terminal (referred to as Command-Line Interface or CLI). Many developers create applications that live in the terminal because, frankly, it's a lot less work. What this means is that small, specialized tools are widely available on Linux since someone likely has already made it.

While there are many, many more commands, this guide will stick to the essentials to get you up and running quickly. The essentials have been divided into the following categories:

- File Operations,
- Permissions,
- Installing Applications,
- Coding,
- Git Essentials, and
- Miscellaneous.

These commands can be accessed in the terminal and can have many different options (called parameters outside of a program, arguments inside a program). Google is your friend here, but an even better friend is `tldr`. It provides simple and concise explanations of commands with examples for the most common uses of those commands.

Learning these commands will take time. It will at times be frustrating. Like anything worthwhile, it takes time, practice, and patience.

Essential Terminal Commands

Most commands have different parameters, some optional and some required. This guide will only cover the most common uses of the most common commands. Use `tldr command` to get more uses for a command. Use `man command` to get detailed information and all parameters for a command.

For most commands, either a full path or relative path may be used for input/output files and directories. Make sure to consider your user access to paths as well. For a command with input and/or output parameters, the input must be a file/directory that your user has access to read. The output must be a file/directory that your user has access to write to.

Most commands also support wildcard characters. Once you're familiar with commands, start experimenting with wildcards. Start by reading more here.

File Operations

cd

`cd` is by far the most commonly used command line utility. This changes what directory you are currently in.

Association: **c**hange **d**irectory

`cd`	Go to your home directory, `/home/username`.
`cd ~`	Go to your home directory, `/home/username`.
`cd /home/username/directory`	Go to a specific directory by providing a full path.
`cd directory`	Go to a directory that is in your current directory.

ls

`ls` is the second most used command. This command will list files and directories in your current location by default, but there are multiple options. If you install and use `zsh` and `oh-my-zsh`, a convenient shortcut is `l`.

Association: **l**i**s**t files

`ls`	Lists files and directories in your current directory
`ls -l`	include more information like ownership and permissions.
`ls -la`	include hidden files and directories.
`ls -la directory`	list all files and information for a specific directory.

mkdir

`mkdir` is used to create a new directory or folder, or many directories at once.

Association: **m**a**k**e **dir**ectory

`mkdir a-directory`	Create a new directory.
`mkdir a-directory b-directory ...`	Make multiple directories.

touch

`touch` has multiple uses. The primary use is to create new file(s). The secondary use is to update the last-accessed timestamp of a file. Note that this command does not create directories, but will update the timestamp of a directory.

Association: **touch**ing a file (to touch means it exists)

`touch file.txt`	Create (or update the access timestamp) a file.
`touch file.txt a-directory ...`	Create/update a file and update a directory.

cp

`cp` copies a file(s)/directory(ies) to a destination directory. This copies the permissions as well as the file contents themselves. `cp` writes over any existing output file as well.

Association: **c**o**p**y a file

`cp input-file output-file`	Copy the input file and create/write the output file. If the output file is a directory, then the input file is copied to that directory
`cp a-input b-input directory`	Copy multiple files into one output directory.
`cp -r input-directory output-directory`	Create a copy of the input directory with the output's name.

mv

`mv` moves a file(s)/directory(ies) to a target destination. This can be used to rename a file or directory, or move file(s)/directory(ies) into another location. If the output file/directory exists, it's contents are written over with the input's content.

Association: **m**o**v**e a file

`mv input-file directory`	Move the input file into the directory.
`mv a-input b-input directory`	Move multiple files into one directory.
`mv input-name output-name`	Rename a directory/file.

rm

`rm` is the delete command, and can remove one or more files at once, and can be used to delete directories as well.

Related: `rmdir`

Association: **r**e**m**ove

There are some important considerations:

- there is no undo button/command. Once you delete something, it's gone forever (unless you have a copy elsewhere).
- If you need to type in your password or use `sudo`, you probably should not `rm` that file(s).
- It is possible for you to "brick" your system by deleting key information, including the root directory `/`.
- A good rule of thumb, if you're not in your home directory (ie. your path does not contain `/home/username`), you shouldn't use `rm`.
- While you're still learning and until you feel comfortable, these are the best guidelines to follow.

`rm input-file`	Remove a file.
`rm a-input b-input`	Remove multiple files at once.
`rm -r a-directory`	Remove a directory. `-r` and `--recurse` are interchangeable.

Permissions

sudo

`sudo` allows a command to be run as the root user (see II - How to use Linux). This is most commonly used in conjunction with your package manager (see Installing Applications), but there are many use cases where you'll want to use it. From editing system configuration files to executing system commands, in Windows terms `sudo` elevates your command to be run as the Administrator.

Related: `su`

Association: **s**witch **u**ser and **do** || **s**uper **u**ser and **do**

`sudo nvim /usr/local/bin/a-custom-script`	Since a regular user doesn't have permission to edit a file in `/usr/local/bin`, switch to the root user to edit the file.
`sudo apt install firefox`	Installing, updating, upgrading, and removing packages requires `sudo` (see Apps).
`sudo cp new-script /usr/local/bin`	Copy a new script to a globally accessible script location (see IV - Making your own CLI tools).

Apps

Package Manager == App Store

A package manager is an app store (in coding, the '==' operation is a boolean expression meaning the left and right are the same value). There are a few important differences though.

- A package manager is "distro-dependent", meaning it is only available to and is designed for specific Linux distributions and families. A package manager can also be available and designed for use for a specific programming language as well (see Setting Up Programming Languages).
    - In everyday use, this means that the command used to install applications can change depending on your Linux distro.
- A package manager is usually a **C**ommand **L**ine **I**nterface, meaning that your interaction with it is limited to the terminal.
- If an application is already installed and you attempt to re-install it, the exact behavior can vary but either the package manager aborts the operation or it re-installs the app.

Raspberry Pi

For the Raspberry Pi, the package manager is **A**dvanced **P**ackage **T**ool, commonly called `apt`. Packages are applications, with all dependencies and instructions already setup.

The `install` and `upgrade` sub-commands will typically require some input from you. This comes in the form of simple `Y` or `N` prompts typically.

`sudo apt update`	Check if any packages have a new version available.
`sudo apt install firefox`	Install the Firefox web browser.
`sudo apt remove firefox`	Uninstall the Firefox web browser.
`sudo apt upgrade`	Upgrade every package installed with `apt` to the latest version.
`apt search firefox`	Find a package based upon search criteria.

Coding

Linux is a programmer's playground. Writing code that will modify your system, create applications, CLI tools, are all easily accessible through Linux. This section will cover how you can write code in the terminal, and how you can run that code.

Text Editor

NeoVim

High-level, Neovim is a customizable and extensible text editor. You can either install a pre-configured Neovim distro (like LunarVim, AstroNvim, or LazyVim) or start from scratch. While you're learning, a pre-configured distro is going to smooth the learning curve and is the general recommendation for starting out with Neovim. It'll include modern conveniences like auto-complete, GitHub Copilot, warnings and errors, and more. Eventually, creating a custom configuration is ideal but ultimately not necessary. The keybindings below are the same across distributions. Additionally, most servers will have at least Vi (an ancestor of Neovim) installed. Part of the power of Neovim (and its ancestors Vi and Vim) is the modal interface. Fancy words aside, this just means that you only operate within different modes. In different modes, the keybindings change. This will only cover the most common and necessary keybinds and commands.

If you want to learn Neovim keybinds, my strongest recommendation is to write 5-8 keybinds on a sticky note with some small notes to remind yourself what it does. Since I'm usually on a laptop, I usually keep it to the right of my trackpad. I still do this regularly as I learn new more powerful keybindings. If you want to make your own configuration file or tweak a distro's configuration, learn the Lua programming language, reference Packer.nvim and Mason.nvim, and checkout this video from the Primeagen.

Personally, I use Neovim with a custom config. Not just in Linux, but MacOS and Windows too. And I use the keybindings in many applications. Neovim has a steep learning curve, and it will feel like learning to type again. Depending on how deep you go and how often you practice, getting the keybindings down will take anywhere from 3-6 months. Proficiency is 1-2 years, while mastery is easily 3-5. It may sound like I'm trying to dissuade you from trying Neovim -I'm not. It is muscle memory for me now, to the point that I use it in every application that supports it. It takes time, effort, and patience, and frankly it isn't for everyone. My biggest, subjective takeaway from Neovim: it makes coding fun, even when the code isn't.

Here are the basic commands. Note that some distributions may change the command. eg. LunarVim is `lvim`.

`nvim`	Open a new file. The file is created with its contents once you save.	Basic file creation.
`nvim file.txt`	Open a file called `file.txt`.	Open a file
`nvim .`	Open the current directory in Neovim, using netrw.	Enter `netrw`
`nvim a-directory`	Open a directory from a path, using netrw.	Enter `netrw`

Keybinds:

Part of the power of Neovim is repeatable commands. The commands below can have a number prefix to indicate how many times to perform that command. Eg. `50j` in command mode translates to moving 50 lines down.

`h`	Move the cursor left a column, but in the same row.	Navigation
`j`	Move the cursor down a row, and attempt to preserve the column.	Navigation
`k`	Move the cursor right a column, but in the same row.	Navigation
`l`	Move the cursor up a row, and attempt to preserve the column.	Navigation
`w`	Move the cursor to the start of the next word or special character, ignoring whitespace.	Navigation
`b`	Move the cursor to the start of the previous word or special character, ignoring whitespace.	Navigation
`i`	Enter `insert mode`, the primary typing mode.	Insert Mode
`I`	Enter `insert mode` at the start of the current line.	Insert Mode
`S`	Enter `insert mode` after clearing the current line.	Insert Mode
`A`	Enter `insert mode` at the end of the line.	Insert Mode
`v`	Enter `visual mode` (highlighting/selecting) at the cursor's current location. Select text with navigation keys.	Visual Mode
`V`	Enter `visual mode` at the cursor's current row. `j` and `k` are used to select more rows.	Visual Mode

`y`	Copy highlighted text to Neovim's clipboard. NOTE: Neovim's clipboard is separate from your system clipboard.	Visual Mode
`"*y`	Copy highlighted text to your system clipboard.	Visual Mode
`u`	Undo.	Special Function
`p`	Paste text from Neovim's clipboard.	Special Function
`:Ex`	Enter Neovim's file explorer, `netrw`.	Enter `netrw`
`%`	Create a file inside the current directory.	`netrw` keybinds
`d`	Create a new directory inside the current directory.	`netrw` keybinds
`D`	Delete a file or empty directory.	`netrw` keybinds
`:w {filename}`	Write (save) the current file. You can specify a filename as well.	File Management
`:q`	Quit (exit without saving) the current file.	File Management
`:wq`	Write (save) and quit (exit) the current file.	File Management
`:%s/original/new/g`	The most common command I use, a highly and easily configurable search and replace. The (optional) `%` means the entire file. By default, it'll only apply to the current line. `original` is the current text and `new` is the replacement. The (optional) `g` means every instance on each line. By default, it'll only apply to the first occurrence.	Advanced Command
`Escape`	Exit current mode.	Exit mode.

Nano - TODO

Setting Up Programming Languages

A broad overview on a handful of languages that you can use with the Raspberry Pi:

Python is the most commonly used language with the Raspberry Pi. Python is known for its syntactic simplicity for novice programmers. Getting started with Python is a fantastic option: but once you're comfortable with foundational programming, learning a language like Java will teach you why Python has some of the limitations it does. The difference between a coder and a programmer is their understanding of why their code works the way it does. Python allows anyone to be up and running, making progress and seeing results immediately. It's a good start, but once you're comfortable with Python you should explore languages like Java or C.

Python

1. Run `sudo apt install python`.
   a. Verify the installation was successful with `python --version`.
   b. Verify that the python package manager was installed with `pip --version`.
2. If you're using a configured text editor like LunarVim, every `.py` file opened will automatically have auto-complete and error flags. Run `nvim hello-world.py`.
   a. Using your text editor of choice, create the following "Hello world" program: `print('hello world!')`
   b. Exit your text editor and run `python hello-world.py`. You should see `hello world!` printed in the terminal.

And that's how to get started coding in Python! To get started using all of those pins on your Raspberry Pi, check out this documentation page on the Raspberry Pi website.

Git Essentials

Git is a critical component of modern software development. It is a Version Control System (VCS), that allows you to work collaboratively and share your projects with other developers across the world. Learning even the bare essentials of Git will massively improve your workflow. Think of Git as Google Docs for developers, except it came out 10 years before Google Docs and Google Docs uses Git or something similar to work.

The high-level concepts of Git involve *repositories*, or *repo* for short. A repo is a directory or folder containing all of the code, installation scripts, and instructions for a piece of software. A repo can be either *local* or *remote*.

A remote repo is stored on a server somewhere, and can be accessed by you, people you specify, or open for the whole internet. A remote repo is generally where the most up-to-date is stored. The most commonly used remote repository is GitHub.

A local repo is stored on your computer. Any changes you make are stored only on your computer, and it can only be accessed by you. The local repository is where a developer does most of their work. Each time you're ready to save the current version of your local repo, you *commit* your changes. This saves the state of every file in the repo, which you go back to and access at any time in the future.

git clone

`git clone` is used to make a local copy of a remote repository on your machine.

`git clone https://github.com/<user>/<repo>`	Create a local copy of the remote repo owned by `<user>` called `<repo>`. This saves the local copy in a new directory in your current directory called `<repo>`.
`git clone <remote> <destination>`	Clone a remote repository and save it in a directory called `<destination>`.

git stage

`git stage` specifies what changes to include in a commit (snapshot) of the repo. This works with the `.gitignore` plain-text file in the repository. This file lists each directory and/or file that will never be included in a commit. `git stage` must be run before `git commit`.

Related: `git add`

`git stage -A`	Include all changes in the commit..
`git stage <file>`	Add a file you want to include in the commit.

git commit

`git commit` creates a snapshot of the repo based upon the changes that are being tracked/included by the `git stage` command.

`git commit`	Open your default text editor to create the commit message. Once you save and quit, a new snapshot of your local repo is saved.
`git commit -m "A commit message."`	Create the snapshot with a commit message.
`git commit --all`	This option auto-stages your changes for the commit, so you do not need `git stage`.

git push

`git push` will attempt to upload your commit(s) to the remote repository.

git pull

`git pull` will download the latest changes from the remote repository.

Quality of Life Improvements

man

`man` is used to open instructional information about a particular command, application, or function. The `man` command is quite powerful, though a touch verbose. It'll provide information on all possible parameters, uses, and idiosyncrasies.

Related: `tldr`

Association: **man**ual

`man <command>`	Open the manual for a command, application, or function. Press `q` to quit, and use `j` and `k` to move down and up respectively.

tldr

`tldr` is the concise version of `man`. Personally, I use `tldr` almost every day. It provides the most common few uses of a command, application, or function.

Related: `man`

Association: **t**oo **l**ong, **d**idn't **r**ead

`tldr <command>`	Get quick information about a command, application, or function.

zsh

`zsh` is a shell, effectively the programming language your terminal uses. `zsh` has more customization options available, such as `oh-my-zsh`, `starship`, and much more. Using your programming skills to customize your operating system is a deep rabbit-hole that you should at least be aware of.

IV - How to unleash the Raspberry Pi’s Potential.

Making your own CLI tools

Scripts & automation

Code writing and text creation

Important rules (syntax)