Intro To GIT SCM

The Power Of Source Control Management

Git SCM

~ Start to Finish

- Overview
- About Version Control
- A Short History of Git
- Installing Git
- First-Time Git Setup
- Getting Help
- Summary

Next Lesson:

- Git Basics

Overview

This lesson is about starting with Git. We will first explain some background regarding version control tools, then move on to how to get Git running on your system and ultimately how to set it up to start working with it. You should know why Git is around at the end of this lesson, why you should use it, and you should all be set up to do that.

About Version Control

~ Start to Finish

- Overview
- Local Version Control
- Centralized Version Control
- Distributed Version Control

About Version Control

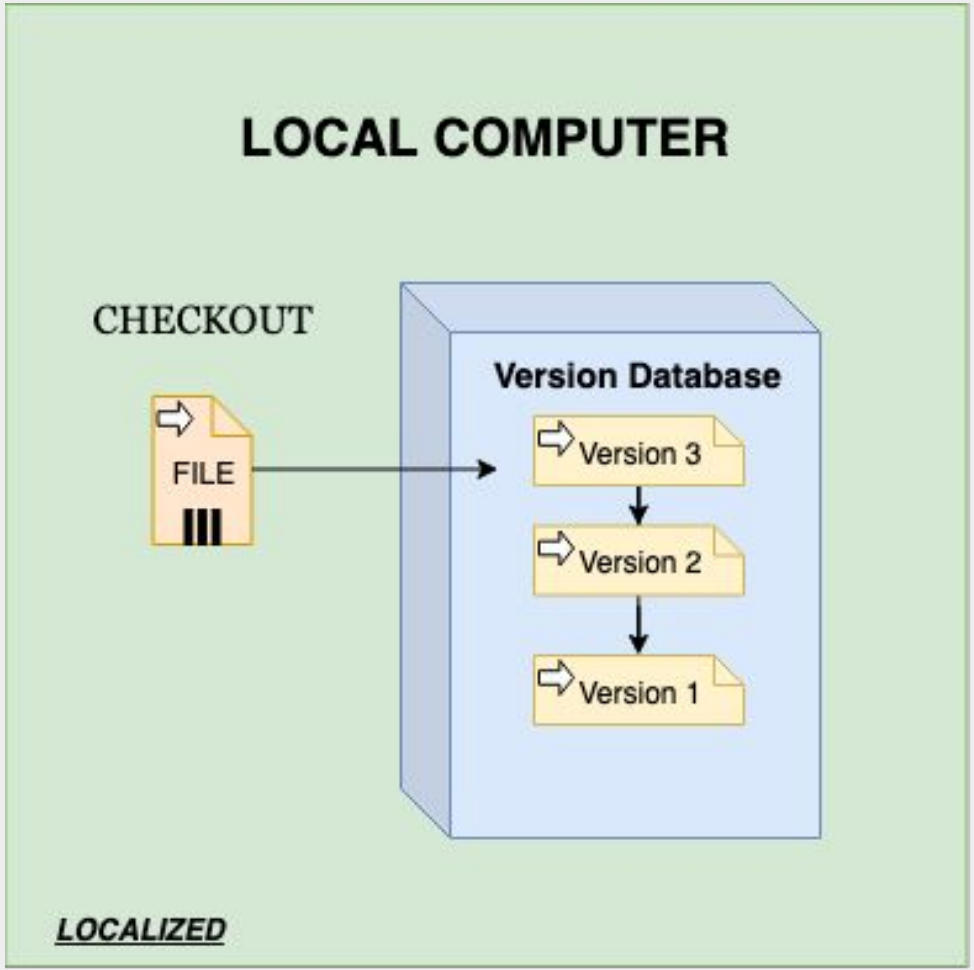
- Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

- Why should I care about version control ?
  - Records Changes.
  - Can be one file or many files.
  - Recall specific versions later.
  - Any file can be managed under Version Control.

Local Version Control

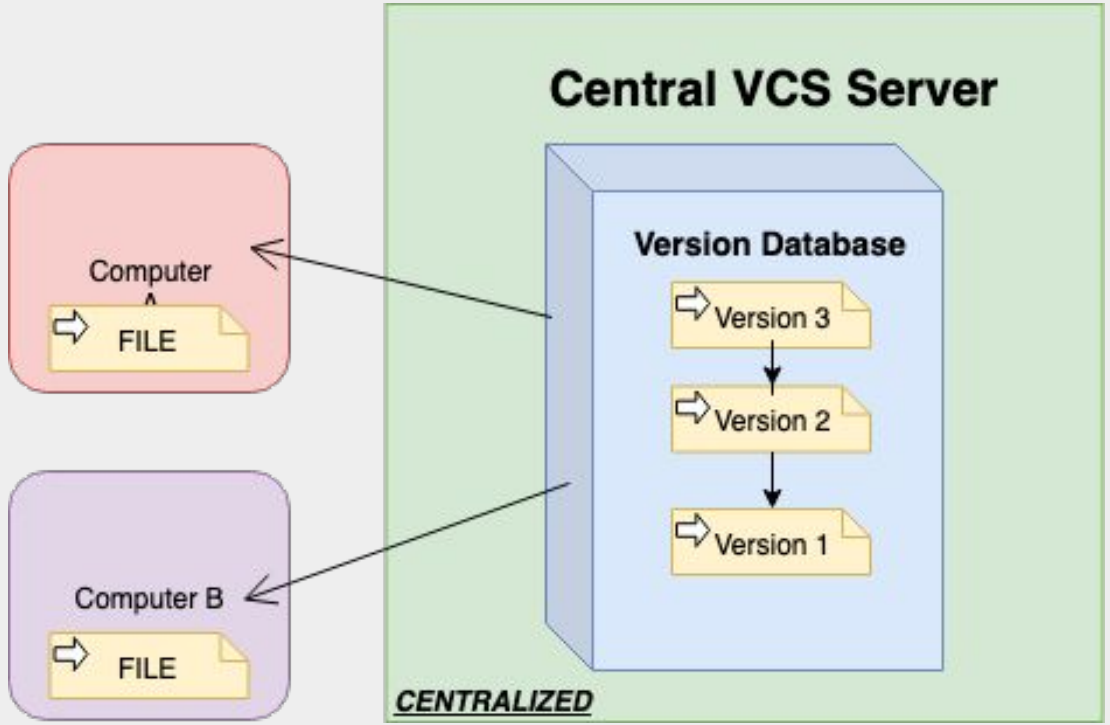
- Many people’s version-control method of choice is to copy files into another directory (perhaps a time-stamped directory, if they’re clever). This approach is very common because it is so simple, but it is also incredibly error prone. It is easy to forget which directory you’re in and accidentally write to the wrong file or copy over files you don’t mean to.

- To deal with this issue, programmers long ago developed local VCSs that had a simple database that kept all the changes to files under revision control.



Centralized Control

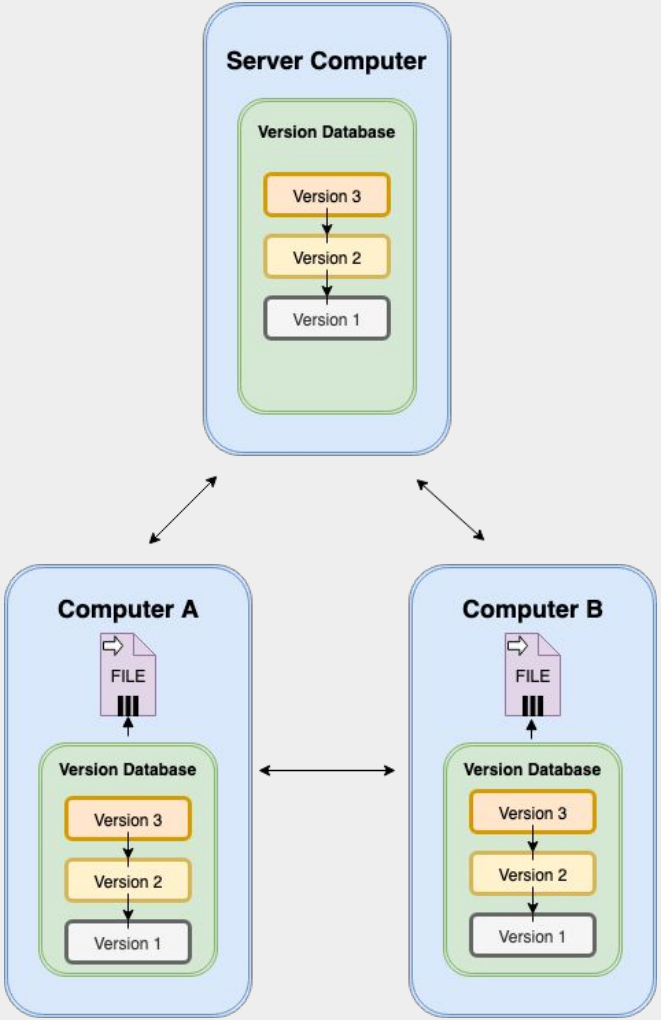
- The next major issue that people encounter is that they need to collaborate with developers on other systems. To deal with this problem, Centralized Version Control Systems (CVCSs) were developed. These systems (such as CVS, Subversion, and Perforce) have a single server that contains all the versioned files, and a number of clients that check out files from that central place. For many years, this has been the standard for version control.



Distributed Version Control Systems

- This is where Distributed Version Control Systems (DVCSs) step in. In a DVCS (such as Git, Mercurial, Bazaar or Darcs), clients don’t just check out the latest snapshot of the files; rather, they fully mirror the repository, including its full history. Thus, if any server dies, and these systems were collaborating via that server, any of the client repositories can be copied back up to the server to restore it. Every clone is really a full backup of all the data.

Distributed



A Short History

~ Start to Finish

- A blast from the past with Git

A Short History

- As with many great things in life, Git began with a bit of creative destruction and fiery controversy.
- The Linux kernel is an open source software project of fairly large scope.
- In 2002, the Linux kernel project began using a proprietary DVCS called BitKeeper.
- In 2005, the relationship broke down, and free-of-charge status was revoked.
- This prompted the Linux development community to develop their own tool based on some of the lessons they learned while using BitKeeper.

- Some of the goals of the new system were as follows:
  - Speed
  - Simple design
  - Strong support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects like the Linux kernel efficiently (speed and data size)
- Since its birth in 2005, Git has evolved and matured to be easy to use and yet retain these initial qualities. It's amazingly fast, it's very efficient with large projects, and it has an incredible branching system for non-linear development

What is Git ?

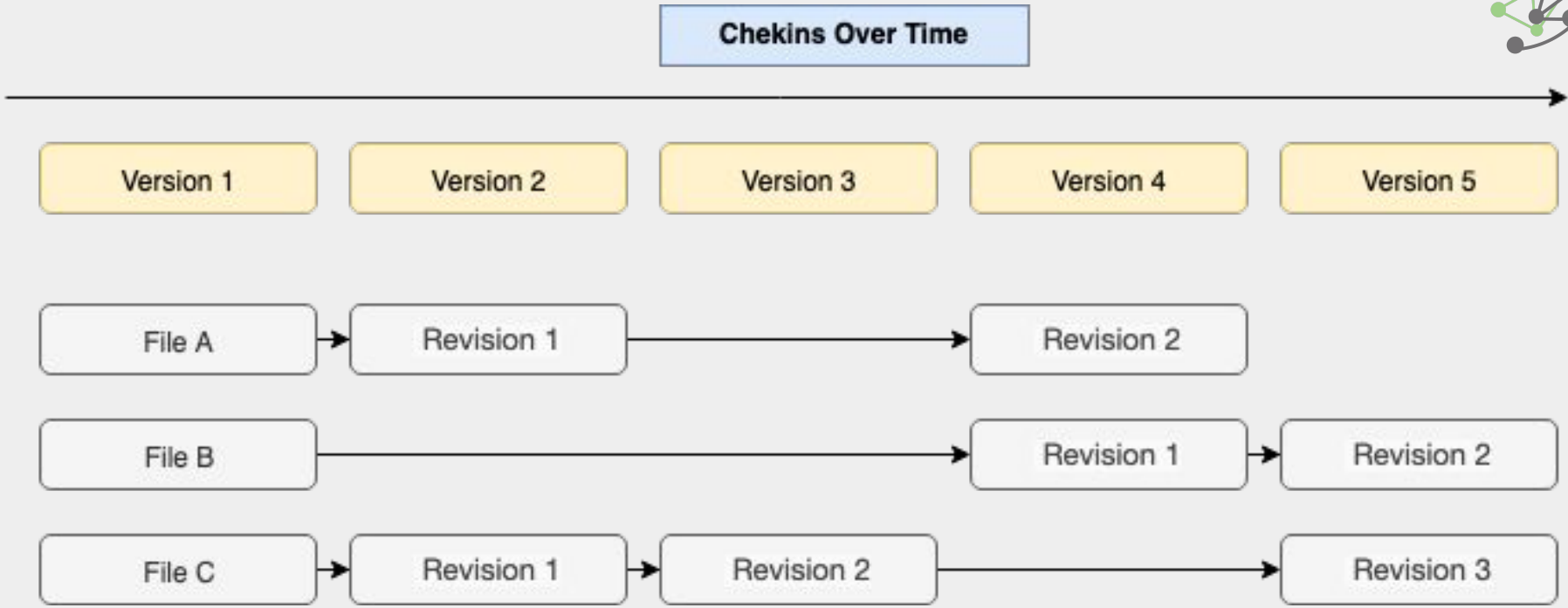
~ Start to Finish

- An understanding of the powerful tool.

Git Overview

What is Git in a nutshell?

- Snapshots, Not Differences
  - Git thinks of its data more like a series of snapshots of a miniature filesystem.
  - Git thinks about its data more like a stream of snapshots
- Nearly Every Operation Is Local
  - Most operations in Git need only local files and resources to operate
- This aspect of Git will make you think that the gods of speed have blessed Git with unworldly powers. Because you have the entire history of the project right there on your local disk, most operations seem almost instantaneous.



Git Overview

Git has Integrity

- Everything in Git is checksummed before it is stored and is then referred to by that checksum. This means it’s impossible to change the contents of any file or directory without Git knowing about it. This functionality is built into Git at the lowest levels and is integral to its philosophy. You can’t lose information in transit or get file corruption without Git being able to detect it.

Git Overview

Git Generally Only Adds Data

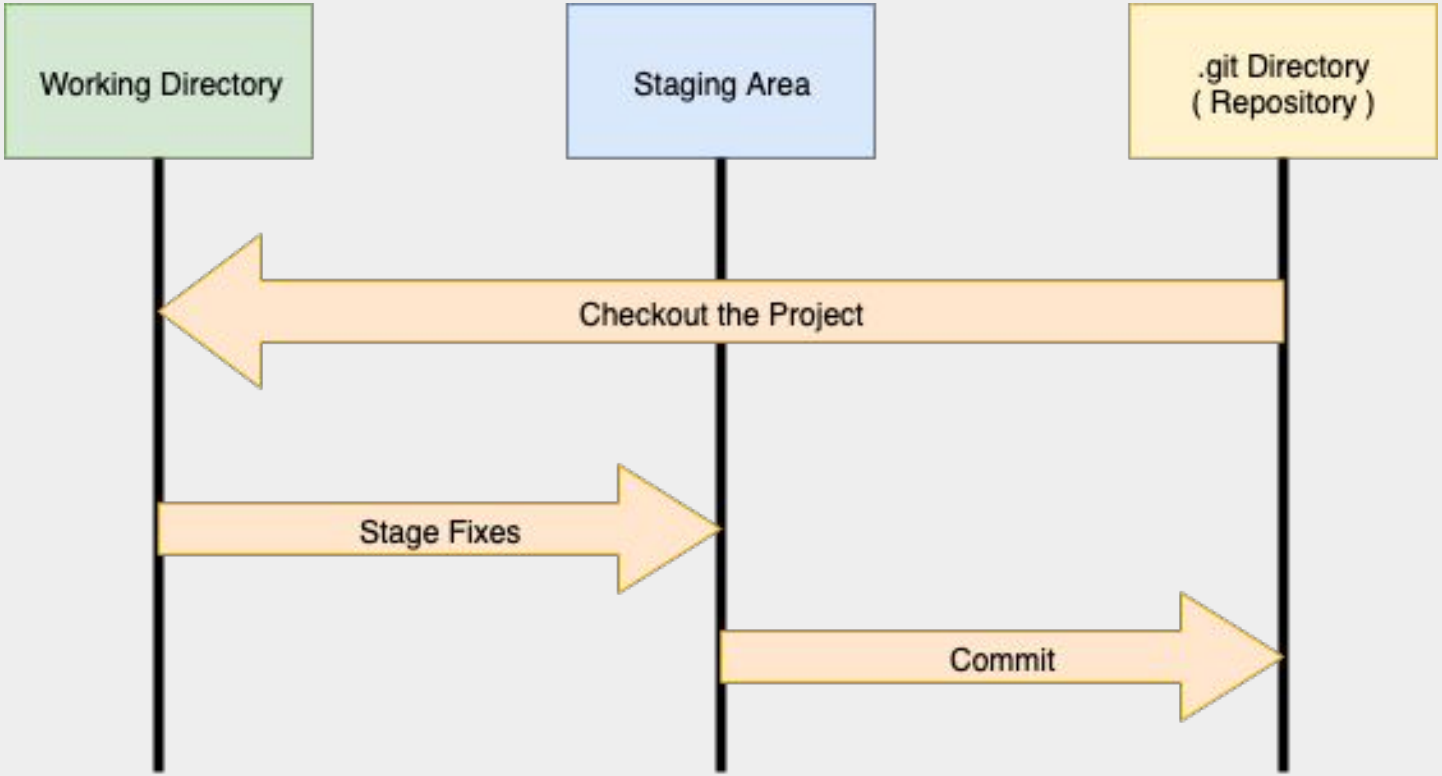
- When you do actions in Git, nearly all of them only add data to the Git database. It is hard to get the system to do anything that is not undoable or to make it erase data in any way

Git Overview

The Three State

- Pay attention now — here is the main thing to remember about Git if you want the rest of your learning process to go smoothly. Git has three main states that your files can reside in: modified, staged, and committed:
  - Modified means that you have changed the file but have not committed it to your database yet.
  - Staged means that you have marked a modified file in its current version to go into your next commit snapshot
  - Committed means that the data is safely stored in your local database.

The Three Stages



The Command Line

- There are a lot of different ways to use Git. There are the original command-line tools, and there are many graphical user interfaces of varying capabilities.
- For one, the command line is the only place you can run all Git commands — most of the GUIs implement only a partial subset of Git functionality for simplicity.
- So we will expect you to know how to open Terminal in macOS or Command Prompt or PowerShell in Windows. If you don’t know what we’re talking about here, you may need to stop and research that quickly so that you can follow the rest of the examples

Command Line Git

- Overview

The Command Line

- There are a lot of different ways to use Git. There are the original command-line tools, and there are many graphical user interfaces of varying capabilities.
- For one, the command line is the only place you can run all Git commands — most of the GUIs implement only a partial subset of Git functionality for simplicity.
- So we will expect you to know how to open Terminal in macOS or Command Prompt or PowerShell in Windows. If you don’t know what we’re talking about here, you may need to stop and research that quickly so that you can follow the rest of the examples

Installing Git

- Installation Prerequisites
- Linux
- macOS
- Windows
- Source

Installation Prerequisites

- Before you start using Git, you have to make it available on your computer.
- Even if it’s already installed, it’s probably a good idea to update to the latest version.
- You can either install it as a package or via another installer, or download the source code and compile it yourself.

Installing on Linux

- If you want to install the basic Git tools on Linux via a binary installer, you can generally do so through the package management tool that comes with your distribution
  - RPM-based distribution, such as RHEL or CentOS), you can use dnf:
    - $ sudo dnf install git-all
  - If you’re on a Debian-based distribution, such as Ubuntu, try apt:
    - $ sudo apt install git-all

Installing on macOS

- There are several ways to install Git on a Mac. The easiest is probably to install the Xcode Command Line Tools. On Mavericks (10.9) or above you can do this simply by trying to run git from the Terminal the very first time.
  - $ git --version
- If you don’t have it installed already, it will prompt you to install it.
- If you want a more up to date version, you can also install it via a binary installer. A macOS Git installer is maintained and available for download at the Git website, at https://git-scm.com/download/mac.

Installing on Windows

- There are also a few ways to install Git on Windows.
- The most official build is available for download on the Git website.
- Just go to https://git-scm.com/download/win and the download will start automatically.

    - Note that this is a project called Git for Windows, which is separate from Git itself; for more information on it, go to https://gitforwindows.org.

Installing from Source

- Some people may instead find it useful to install Git from source, because you'll get the most recent version.
- If you do want to install Git from source, you need to have the following libraries that Git depends on: autotools, curl, zlib, openssl, expat, and libiconv.
- You can use one of these commands to install the minimal dependencies for compiling and installing the Git binaries:
    - $ sudo dnf install dh-autoreconf curl-devel expat-devel gettext-devel \ openssl-devel perl-devel zlib-devel
    - $ sudo apt-get install dh-autoreconf libcurl4-gnutls-dev libexpat1-dev \ gettext libz-dev libssl-dev

Installing from Source continued….

- In order to be able to add the documentation additional dependencies are required. You can install them using these commands.
  - $ sudo dnf install asciidoc xmlto docbook2X
  - $ sudo apt-get install asciidoc xmlto docbook2x
- If you’re using a Debian-based distribution (Debian/Ubuntu/Ubuntu-derivatives), you also need the install-info package:
  - $ sudo apt-get install install-info
- If you’re using a RPM-based distribution (Fedora/RHEL/RHEL-derivatives), you also need the getopt package (which is already installed on a Debian-based distro):If you’re using a RPM-based distribution (Fedora/RHEL/RHEL-derivatives), you also need the getopt package (which is already installed on a Debian-based distro):
  - $ sudo dnf install getopt
  - $ sudo apt-get install getopt

Installing from Source continued….

- Additionally, if you're using Fedora/RHEL/RHEL-derivatives, you need to do this due to binary name differences.
  - $ sudo ln -s /usr/bin/db2x_docbook2texi /usr/bin/docbook2x-texi
- When you have all the necessary dependencies, you can go ahead and grab the latest tagged release tarball from several places.
- You can get it via the kernel.org site, at https://www.kernel.org/pub/software/scm/git , or the mirror on the GitHub website, at https://github.com/git/git/releases.
- It's generally a little clearer what the latest version is on the GitHub page, but the kernel.org page also has release signatures if you want to verify your download.

Compiling Git source and Updates

- To, compile and install:
  - $ tar -zxf git-2.0.0.tar.gz
  - $ cd git-2.0.0
  - $ make configure
  - $ ./configure --prefix=/usr
  - $ make all doc info
  - $ sudo make install install-doc install-html install-info

- After this is done, you can also get Git via Git itself for updates:
  - $ git clone git://git.kernel.org/pub/scm/git/git.git

First Time Setup

- Understanding
- Your Identity
- Your Editor
- Check Your Settings

First time setup

- Now that you have Git on your system, you'll want to do a few things to customize your Git environment.
- You should only have to do this once.
- They normally will not change if you upgrade git

Understanding how to make changes

- Git comes with a tool called git config that lets you get and set configuration variables that control all aspects of how Git looks and operates. These variables can be stored in three different places:
    - /etc/gitconfig file:
    - ~/.gitconfig or ~/.config/git/config file:
    - config file in the Git directory
- Each level overrides values in the previous level, so values in .git/config trump those in /etc/gitconfig.

Using the /etc/gitconfig file:

- This file contains values applied to every user on the system and all their repositories. If you pass the option --system to git config, it reads and writes from this file specifically. When using a system configuration file, you need administrative or superuser privilege to make changes to it.

Using the ~/.gitconfig or ~/.config/git/config file:

- When using this file, all values are specific personally to you, the user. You can make Git read and write to this file specifically by passing the --global option, and this affects all of the repositories you work with on your system.

Using the config file in Git’s directory

- The config file in the Git directory (.git/config) of whatever repository you’re currently using
- File is specific to that single repository.
- You can force Git to read from and write to this file with the --local option, but that is in fact the default.
- You need to be located somewhere in a Git repository for this option to work properly.

Where does Git look?

- To find out where Git is looking to retrieve these credentials use the following command:
  - $ git config --list --show-origin
- Once you know where these credentials are being stored, make the necessary changes and save the file.
- This will ensure

Your Passport

- The first thing you should do when you install Git is to set your user name and email address. This is important because every Git commit uses this information, and it’s immutably baked into the commits you start creating:
  - $ git config --global user.name "John Doe"
  - $ git config --global user.email johndoe@example.com
- Again, you need to do this only once if you pass the --global option, because then Git will always use that information for anything you do on that system. If you want to override this with a different name or email address for specific projects, you can run the command without the --global option when you’re in that project.
- Many of the GUI tools will help you do this when you first run them.

Your Editor or IDE

- Now that your identity is set up, you can configure the default text editor that will be used when Git needs you to type in a message. If not configured, Git uses your system's default editor. If you want to use a different text editor, such as Emacs, you can do the following:
  - $ git config --global core.editor emacs
- On a Windows system, if you want to use a different text editor, you must specify the full path to its executable file. This can be different depending on how your editor is packaged. In the case of Notepad++, a popular programming editor, you are likely to want to use the 32-bit version, since at the time of writing the 64-bit version doesn't support all plug-ins. If you are on a 32-bit Windows system, or you have a 64-bit editor on a 64-bit system, you'll type something like this:
  - $ git config --global core.editor "'C:/Program Files/Notepad++/notepad++.exe'
  - -multiInst -notabbar -nosession -noPlugin"

Check your Settings

- If you want to check your configuration settings, you can use the git config --list command to list all the settings Git can find at that point:
  - $ git config --list

- You can also check what Git thinks a specific key’s value is by typing git config <key>:
  - $ git config user.name

Help !!!!

I really need help…
Is there a Git 119 ?

- Getting Help
- A Simple Example
- Accessing Help Commands

Getting Help

- If you ever need help while using Git, there are three equivalent ways to get the comprehensive manual page (manpage) help for any of the Git commands:

    - $ git help <verb>
    - $ git <verb> --help
    - $ man git-<verb>

A Simple Example

- For example, you can get the manpage help for the git config command by running
    - $ git help config
- These commands are nice because you can access them anywhere, even offline.

Help Commands

- In addition, if you don’t need the full-blown manpage help, but just need a quick refresher on the available options for a Git command, you can ask for the more concise “help” output with the -h or --help options, as in:

```
$ git add -h
usage: git add [<options>] [--] <pathspec>...

    -n, --dry-run         dry run
    -v, --verbose         be verbose

    -i, --interactive     interactive picking
    -p, --patch           select hunks interactively
    -e, --edit            edit current diff and apply
    -f, --force           allow adding otherwise ignored files
    -u, --update          update tracked files
    --renormalize         renormalize EOL of tracked files (implies -u)
    -N, --intent-to-add   record only the fact that the path will be added later
    -A, --all             add changes from all tracked and untracked files
    --ignore-removal      ignore paths removed in the working tree (same as --no-all)
    --refresh             don't add, only refresh the index
    --ignore-errors       just skip files which cannot be added because of errors
    --ignore-missing      check if - even missing - files are ignored in dry run
    --chmod (+|-)x        override the executable bit of the listed files
```

Summary

- You should have a basic understanding of what Git is and how it’s different from any centralized version control systems you may have been using previously.
- You should also now have a working version of Git on your system that’s set up with your personal identity.
- It’s now time to learn some Git basics.

All Git’s Info can be found at: https://git-scm.com/doc