# Introduction

Git is a software tool used for Source Code Management also known as SCM. Source Code Management software allows a team of programmers to share set of code that they are all working on. The SCM handles keeping track of changes and preventing a set of changes by one programmer to overwrite the changes of another.

# Terms

## Repositories

Repositories are the heart of how git manages source control. There are two types of repositories used in a project. The first is the remote repository. This is where the primary source code resides. This is the master code. Our remote repositories are on the GitHub.com and is accessed through the git server software running on the GitHub computers.

The other type of repository is their local repository. Each developer or programmer creates one or more local repositories on their personal computer/laptop. Each local repository is initially populated from the remote repository using the ‘git clone’ command, more about ‘git clone’ later. All new code and modified code is done in local repositories. All changes to the source code are tested and debugged on the programmers’ laptop. The programmer can commit changes to their local repository as they are working on changes. Once the new and modified code has been tested and verified and committed to the local repository, the changes are then merged with the remote repository. Once the new code has been merged with the remote repository, it becomes available to the other programmers on the team.

## Branches

Branches are versions of the source code that can be developed independently of other branches. All repositories have a main branch. Other branches can be created as needed for other subprojects or for bug fixes.

In our current SCAROB repository, we are only using a single main branch. This section will be fleshed out when we need to branch development.

# Git Commands

## git status

The *git status* is probably the most commonly used git command. It provides the status of your local repository. It will list which files have been added or modified, which files have been marked for committing, and other useful information.

|  |
| --- |
| >git status |

## git add

The *git add* command is used to mark a modified file or a new file so that it can be committed to the local repository. Some GUI applications hide this command and performs it in the background when a commit is run.

|  |
| --- |
| >git add <list of files to be committed>  example:  >git add README.md doc/Doxyfile |

A directory or folder cannot be added directly to a repository. Adding and committing a file in the directory or folder will automatically create the directory or folder in the repository.

## git commit

The *git commit* command is how changes are stored in the local repository. In some SCM applications, a commit is called a checkin. In terms of SCM, the two keywords mean the same thing. With git, we use the term commit because it matches the git command to move changed code into the local repository.

The commit process takes more than running the command. As the git commit command runs, an editor pops up. You enter a descriptive and useful comment that explains what changes were made and why. Make sure that these comments are meaningful to the other developers on the team.

A comment of “fixed bug” is meaningless. It does not describe what was done or why. A comment like “fixed bug ### that caused the motors to stop at random times.” is meaningful.

|  |
| --- |
| >git commit |

### Selecting Commit Editor

The default editor that is used for the commit is set when Git is installed. This may not necessarily be the editor you want to use. The default editor can be set at a later time by running the following command:

|  |
| --- |
| >git config –global core.editor <editor-command> |

The <editor-command> is usually just the editor name. It can contain options for the editor.

The editor used as the default editor must be a text[[1]](#footnote-2) editor and installed on the computer that the local repository is on.

## git push

Once changes to the source code has been verified, tested, and are working, these changes can be moved to the remote repository on GitHub to make them available for the rest of the programming team. This is done using the *git push* command.

Sometimes, the push command will fail if there are conflicts with the changes. For example, if two programmers change the same section of code. The first programming to push those changes to the remote repository will be successful. When the second programmer attempts to push their changes to the remote repository, git will not know how to add these changes. In this case, the second programmer will need to resolve the merge issues manually. The code that conflicts will be highlighted in the local repository by having both sets of code enclosed in “============”, “>>>>>>>>>>>>>>>”, and “<<<<<<<<<<” lines. Once the merger issues have been resolved, the new code can be committed to the local repository and then pushed to the remote repository.

|  |
| --- |
| >git push |

### Authentication

Some times the git push will fail because the user is not or no longer recognized as a valid programmer for the remote repository. When this happens, you need to reauthenticate yourself using the GitHub CLI command *gh auth login* from a command prompt window or a terminal on your laptop and the GitHub mobile app on your phone. The gh command will tell you what to do: log into your GitHub Account on a browser and the browser will ask you to enter a code number into the GitHub app on your phone.

|  |
| --- |
| >gh auth login |

## git pull

As other developers push changed code to the remote repository, your local repository becomes out of sync. To restore your local repository so that you are using the latest version of the code, you need to fetch those changes. The *git pull* command is how you make your local repository up to date.

If you have a file that is modified, git will try to merge the changes from the remote repository into your modified file. If it cannot, you end up with merge issues. Just like merge issues resulting from a *git push*, you need to manually resolve those changes locally. The difference between a pull and a push merge issue, you do not push your resolved merge issues until you are ready for your next *git push*.

|  |
| --- |
| >git pull |

## git clone

The *git clone* command is used to create a local repository. This command needs to be run in the folder that is to the parent folder to the local repository. The remote repository URL can be obtained from the GitHub page for that repository. It is available in the green button on the main page for the source code. Copy and paste the HTTPS version of the Repository URL. The clone command will create a local repository in the subfolder specified in the command as <folder-name>.

|  |
| --- |
| >git clone <remote-repository-URL> <folder-name>  example:  >git clone <https://github.com/Star-City-Robotics/SwerveDrive-2024.git>SwerveDrive |

Note, if you do not specify the local repository folder name, the default is the name of the remote repository. In the example above, that would be ***SwerveDrive-2024***.

# Useful Resources

* Git Version Control Introduction:
  + <https://docs.wpilib.org/en/stable/docs/software/basic-programming/git-getting-started.html>

1. A text editor is an editor that only places the characters that you type into the file. Some text editors are: emacs, vim, note, note++. Editors like Word are not text editors because they add hidden characters that allow styling to be added to the text, styling like bold, italic, color. [↑](#footnote-ref-2)