Git Basics Tutorial

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

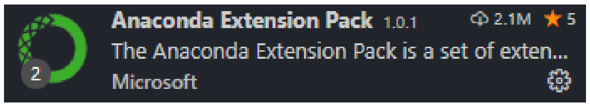
Source: https://xkcd.com/1597/

Contents

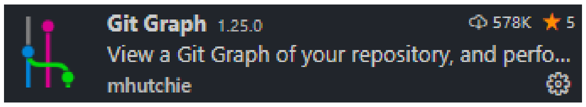
Installation and Setup

To follow along with this tutorial, you will need:

- The VSCode IDE
    - It is strongly recommended that you use VSCode because of the built-in Git services. It will also be easier to follow along with the tutorial. However, other IDEs will still work.
    - If using VSCode, ensure you install the "Anaconda Extension Pack" for compatibility with the Python language and the "Git Graph" extension so you can visualize your git tree:

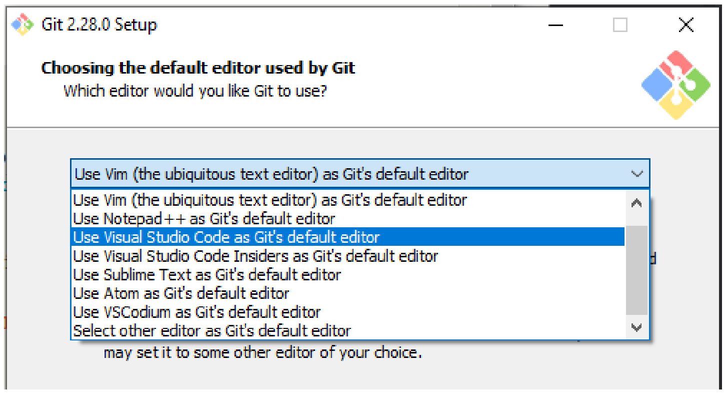


Anaconda Extension Pack 1.0.1 2.1M ★5
The Anaconda Extension Pack is a set of exten...
Microsoft



Git Graph 1.25.0 578K ★5
View a Git Graph of your repository, and perfo...
mhutchie

- An installation of Git
    - **Important note:** When installing Git, ensure you pick your personal IDE as the default editor. The standard option, Vim, is not beginner friendly.



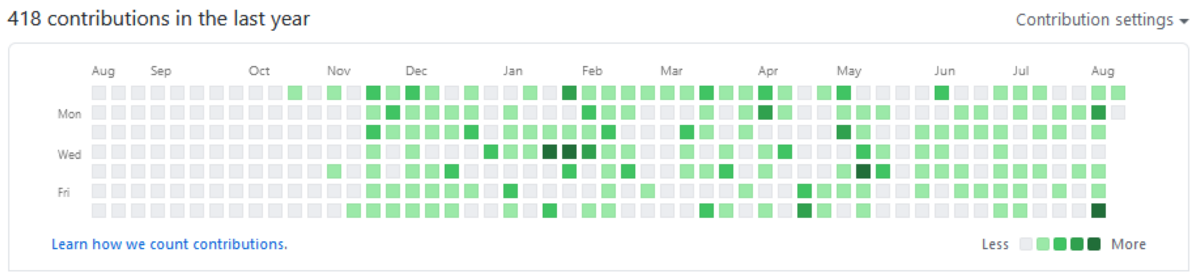
Git 2.28.0 Setup

Choosing the default editor used by Git
Which editor would you like Git to use?

Use Vim (the ubiquitous text editor) as Git's default editor
- Use Vim (the ubiquitous text editor) as Git's default editor
- Use Notepad++ as Git's default editor
- Use Visual Studio Code as Git's default editor
- Use Visual Studio Code Insiders as Git's default editor
- Use Sublime Text as Git's default editor
- Use Atom as Git's default editor
- Use VSCodium as Git's default editor
- Select other editor as Git's default editor

may set it to some other editor of your choice.

    - If Git was installed correctly, entering "git --version" into your terminal should work without error. To link your computer to your GitHub account, use the following commands (swap [name] and [github-email] with your information):
        - `git config --global user.name "[name]"`
        - `git config --global user.email "[github-email]"`
- A GitHub account (Get a Pro account and other goodies with the Student Developer Pack)
    - If you are a software student, having a GitHub account can be a great way to show potential employers that you are comfortable with Git and develop software regularly. Many students put their GitHub account on their resume as a portfolio of software experience. The number of commits you make to a repository in a particular day is also tracked by GitHub and is visible to the public. In order to get more experience with using Git (and to fill out your activity calendar) it is recommended that you make a repo for each of your software classes and commit your progress towards assignments.



418 contributions in the last year

Background

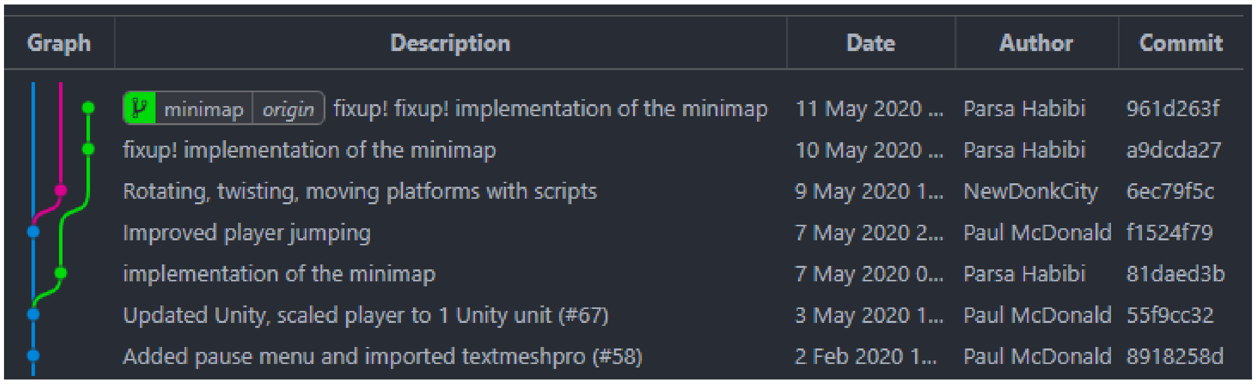
What is Git?

Largescale software projects involving numerous developers working on the same source code is commonplace in the software industry. Git is a version control tracking tool that makes this kind of collaborative software development possible by allowing developers to work in parallel without breaking the project for everyone else. It also allows developers to instantly jump back to previous versions of a project to determine when a bug was introduced into the codebase.

Using Git involves syncing a local project on your computer (local repository) to a project hosted on the internet (online repository / origin) via Git hosting services such as GitHub or BitBucket. Your local repository is a clone of the online repository that you publish changes to. Once changes have been made locally, they are "pushed" to the online repository for safe keeping. If you are working with a group, updates that have been "pushed" to the online repository can be "pulled" by other developers to their local repositories such that they can work off of your updates. Git is considered "distributed/decentralized" source control because a copy of the project repository is located on each developer's computer in addition to the copy hosted online. As such, work can be performed by each developer offline and then merged into the shared online repository.

What is a commit?

Commits are the foundation of using Git. Software version history is tracked using trees of commits (i.e. formally published updates). Each commit represents a version of the software project and has a corresponding description, date, author, and hash identification number. A small portion of a commit tree, also known as a Git Graph, can be seen in the image below.



Graph	Description	Date	Author	Commit
	minimap *origin* fixup! fixup! implementation of the minimap	11 May 2020 ...	Parsa Habibi	961d263f
	fixup! implementation of the minimap	10 May 2020 ...	Parsa Habibi	a9dcda27
	Rotating, twisting, moving platforms with scripts	9 May 2020 1...	NewDonkCity	6ec79f5c
	Improved player jumping	7 May 2020 2...	Paul McDonald	f1524f79
	implementation of the minimap	7 May 2020 0...	Parsa Habibi	81daed3b
	Updated Unity, scaled player to 1 Unity unit (#67)	3 May 2020 1...	Paul McDonald	55f9cc32
	Added pause menu and imported textmeshpro (#58)	2 Feb 2020 1...	Paul McDonald	8918258d

When you edit a software file and press save, that change is not recorded by version control until it has been committed. In order to commit changes (i.e. publish changes into version control history), a few general steps must be followed:

1. Changes must be made to the local repository.
2. Desired changes must be staged (i.e. selected for committal).
3. Staged changes must be committed to the local repository (i.e. published into local history).
4. Locally committed changes must be "pushed" to the online repository (i.e. published into online history).

In a sense, managing changes between your local and online repos is similar to sending supplies between two space stations. On your local station, you can't just throw things into the vacuum of space and expect them to be picked up by the other station. You must organize what you want to send into a pod that is accepted by the other station. With Git, you must stage your changes by declaring which updates you want to be published (i.e. load the pod with the proper supplies). Then, you have to commit those changes by formally publishing them into your source control tree (i.e. close the pod and ready it for transfer). Once satisfied, you must then "push" your published changes to the online repository (i.e. launch the pod to the other station).

Git workflow

As seen in previous image, a commit tree typically consists of multiple branches. The main project resides in the "master" branch. Typically, when implementing a new feature, a developer creates a new branch on the commit tree such that their changes do not break the working code that other developers rely on. When they have finished their update, they merge their branch back into the master branch such that the main project contains the new feature. This can be seen in the previous image. A branch called "minimap" (shown in green) was created that runs in parallel with the master branch (shown in blue). On that branch, the developer has published 3 commits towards a new feature. Since they are working on a parallel version of the project, they are able to implement their changes and debug their feature without breaking the source code on the master branch. Note that the "minimap" branch icon says "origin" because it corresponds with the online repository.

Git is useful for working with other developers in parallel, however, it is also handy for a solo developer. For example, cloning a project to multiple computers allows a single developer to work in multiple locations without taking up space on a cloud drive. In the case that a bug was found in the code, a solo developer can incrementally navigate back to previous commits (i.e. previous versions of the project) to find out which changes introduced the bug. For students working alone on a simple project, such as an assignment, the use of branches may not be necessary since all work is done in sequence.

Using Git in Terminal VS GUI

Git can be used through terminal commands or through GUIs. Although initially easier to learn, using Git through a GUI will hold you back. Git GUIs only provide a fraction of the functionality that can be achieved through the terminal. However, it is usually best to find a balance. Using the built-in Git GUI functionalities in a modern IDE such as VSCode in conjunction with the terminal is what works best for most people. It is recommended that you become comfortable with using both rather than exclusively using one.

Basic Terminal Navigation

To follow along with this tutorial, you will need to know the basics of navigating a terminal. If you do not know how, check the following tutorials:

- Linux/Mac: https://www.youtube.com/watch?v=j6vKLJxAKfw
- Windows: https://www.youtube.com/watch?v=MBBWVgE0ewk

Additional Theory

The above theory should be sufficient to take on this tutorial. However, if you are interested in learning more, the following links should be helpful:

- Learn Git in 15 Minutes Video: https://www.youtube.com/watch?v=USjZcfj8yxE
- Git handbook: https://guides.github.com/introduction/git-handbook/
- GitHub tutorials: https://guides.github.com/activities/hello-world/
- GitHub Training & Guides YouTube: https://www.youtube.com/githubguides
- Online interactive visual tutorial: https://learngitbranching.js.org/

Tutorial Content

Since Git is such a robust tool, there is much more to learn. However, Git is best learned through example. The rest of the tutorial will walk you through some fundamental Git use cases including:

- Creating/Cloning a Git project
- Committing Project Changes
- Branching and Merge Conflicts

Keep in mind that this tutorial only focuses on fundamentals and does not handle more advanced, yet common concepts such as:

- Intermediate
  - Stashing (Temporarily storing uncommitted changes)
  - Resetting (Resetting the local branch to match the online branch)
  - Pull requests (Requesting a review of your code before it can be merged to master)
- Advanced
  - Rebasing (Moving the base of a branch to a different commit)
  - Interactive rebasing (Rearranging/rewriting the git tree)
  - Force pushing and associated risks (Overwriting online history with local changes)

Now that the basic theory is out of the way, let’s start by learning how to begin work in a Git project.

Part 1 – Creating/Cloning a Git project

There are two ways to begin using Git:

- Adding Git version control to an existing software project
- Cloning an existing Git project to your local computer

In this example, 1 person (Person A) will create and initialize the Git repository and the other 2 (Person B and Person C) will clone a copy of the repository to their computers. Proceed with the following steps and, although these steps are split between people, make sure that everyone is learning each process.

Note: If you do not have a partner, perform the work of both Person A and Person B. When you clone the repository for Person B, make sure you perform all tasks allocated to Person B in that repository.

Initial setup (Person A)

1. Create the local repository
    a. Download the starting `Converter.py` file and save it within a project folder on your computer.
    b. Open a new terminal and **navigate to the local project folder directory**.
    c. Make an initial commit to publish the project into source control history using the following commands (**Note:** We will cover commits in more detail in the next section. Don't worry about understanding the commands for now.)
        i. `git init`
        ii. `git add .` (don't forget to include the period)
        iii. `git commit -m “Initial commit”`
        iv. `git push origin master`
2. Create the online repository
    a. Login to GitHub and begin creating a new repository.
    b. Name your repository “Project_A_B_C” but replace “A”, “B” and “C” with the first names of each group member.
    c. Copy the repository's Git link to your clipboard. This hyperlink is used to connect the online repo to a local repository on your computer.
3. Link local and online repositories
    a. **In a terminal on the local project folder directory**, connect the repositories using the following commands:
        i. `git remote add origin [Git-link]` (insert your Git hyperlink)
        ii. `git push origin master`
4. Add group members as contributors to the GitHub repo
    a. From your repo page, navigate to Settings > Manage access > Invite a collaborator
    b. Add your group members by username or email.

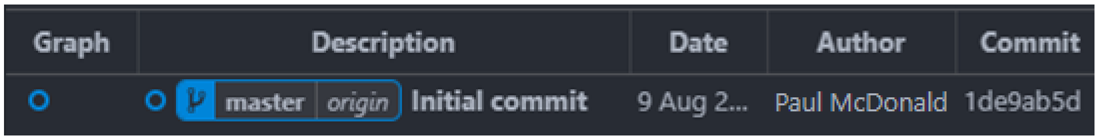
Cloning (Person B and C)

1. Check your email and follow the corresponding link to be added to the repo.
2. Navigate to the repository page and find the Git link.
3. Clone the online repository to your local computer.

        a. Open a new terminal and **navigate to the directory that you want your project folder to be placed**.
        b. Clone the repository using the following command:
            i. `git clone [Git-link]` (insert your Git hyperlink)
4. Open the newly cloned Git project in your IDE. You should see the file that Person A committed.

Part 2 - Committing Project Changes

Now that your group has access to the Git project, it's time to learn how to commit (i.e. publish changes into version control history). Currently, there is only one commit in the repo. Person A published an initial commit when setting up the repositories. As such, your git tree looks like the image below. If using VSCode, to see your Git tree, click the "Git Graph" button on the bottom bar of your VSCode window. If your IDE does not have built-in Git functionality, you can find the Git Graph in your GitHub repository page by navigating to Insights > Network.

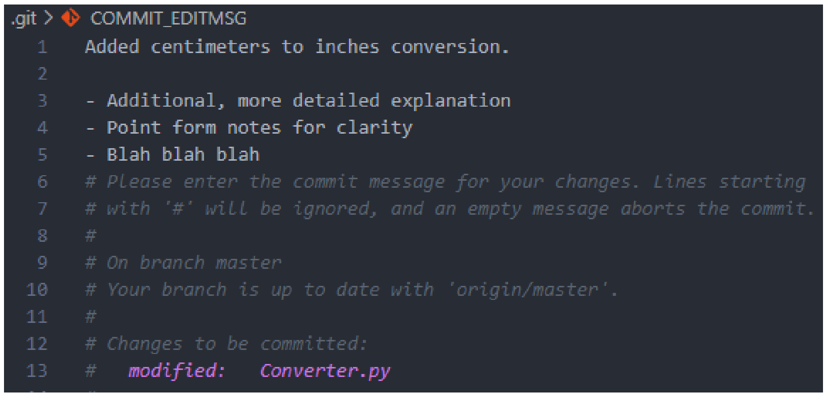


Graph	Description	Date	Author	Commit
○	○ master origin Initial commit	9 Aug 2...	Paul McDonald	1de9ab5d

Currently, it's not much of a tree because it only consists of one node (i.e. one commit). However, in this section, each group member will be submitting one additional commit. However, since we have not yet covered branching, your group cannot work in parallel without creating merge conflicts. This concept will be discussed more in a later section. The following steps must be taken by each group member **one at a time**.

To commit an update to the project:

1. Pull the most recent changes from the online repository:
    a. **In a terminal on the local project folder directory**, use the following command:
        i. `git pull` (if applicable, pull previous member's changes into your local repo)
            - **Note:** `git pull` is a combination of the commands: `git fetch` and `git merge`. It fetches (i.e. checks the online for repo for any recent changes) and then merges changes from the online repo into your local repo.
2. Implement your assigned changes to the project's `.py` file and save.
3. Commit your changes locally
    a. **In a terminal on the local project folder directory**, use the following commands:
        i. `git add .` (period signifies to stage *all* changes in the current directory)
        ii. `git commit` (commit staged changes)
    b. Assuming your default editor is properly assigned, a file will open in your IDE that will allow you to enter a commit title and description. An example commit title and description is shown below. Ensure to include one line of white space between your title and description.



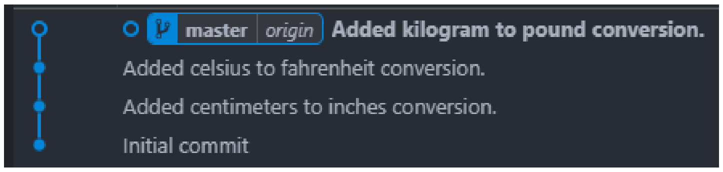
```
.git > COMMIT_EDITMSG
Added centimeters to inches conversion.

- Additional, more detailed explanation
- Point form notes for clarity
- Blah blah blah
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Your branch is up to date with 'origin/master'.
#
# Changes to be committed:
#   modified:   Converter.py
```

   c. Once done, save and close the file.

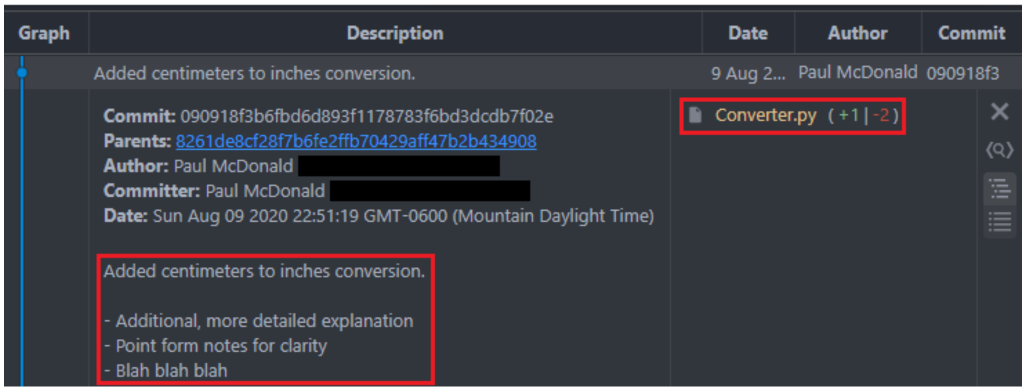
4. Push your commits to the online repository
   a. **In a terminal on the local project folder directory**, use the following command:
      i. `git push`
5. If all group members have not made a commit, select the next person to follow these steps.

After each group member has gone, your tree should look more like image below.



```
master  origin  Added kilogram to pound conversion.
Added celsius to fahrenheit conversion.
Added centimeters to inches conversion.
Initial commit
```

Note that, if you added a more detailed description to your commit, it is reflected in the commit tree.



Graph	Description	Date	Author	Commit
	Added centimeters to inches conversion.	9 Aug 2...	Paul McDonald	090918f3

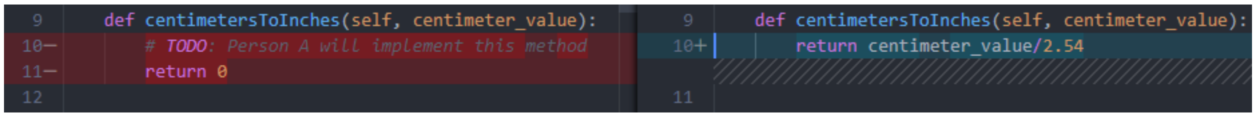
```
Commit: 090918f3b6fbd6d893f1178783f6bd3dcdb7f02e
Parents: 8261de8cf28f7b6fe2ffb70429aff47b2b434908
Author: Paul McDonald [redacted]
Committer: Paul McDonald [redacted]
Date: Sun Aug 09 2020 22:51:19 GMT-0600 (Mountain Daylight Time)

Added centimeters to inches conversion.

- Additional, more detailed explanation
- Point form notes for clarity
- Blah blah blah
```

Converter.py (+1 | -2)

Additionally, the files that were changed in the commit are indicated on the right. As displayed, one line of code was added and two lines of code were removed. Clicking the changed file will open it and highlight the changes introduced by the commit as seen below.



```
 9    def centimetersToInches(self, centimeter_value):
10-       # TODO: Person A will implement this method
11-       return 0
12
```

```
 9    def centimetersToInches(self, centimeter_value):
10+       return centimeter_value/2.54
11
```

Part 3 - Branching and Merge Conflicts

Now that you know how to make basic commits, it's time to learn how to branch so that you and your group members can work in parallel rather than in sequence. Before you do, you should know about the following commands. If you ever need to check which branch you are on, what files you have staged for committal, or whether a merge is currently in progress, use `git status`. If you accidentally mess up your local branch and want to reset it to match the corresponding online branch, use `git reset --hard origin/[branch name]`. **Ensure you are on the correct branch if running this command.**

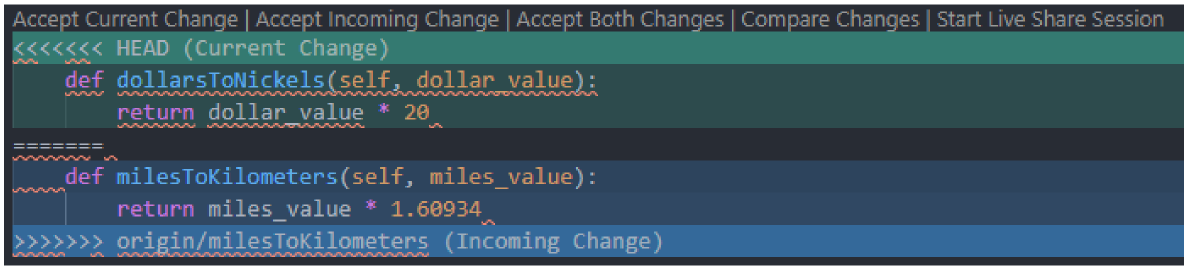
This time, each person will make their own branch and implement a new conversion of their choice. Once each person has committed the desired changes to their branch, they will merge their branch back into the master branch. The steps that each group member should perform are as follows:

1. Create and checkout a new branch.
    a. **In a terminal on the local project folder directory**, use the following commands:
        i. `git branch [branch name]` (replace [branch name] with a descriptive name that describes the work that will be done)
        ii. `git checkout [branch name]` (change working branch to new branch)
2. Make desired changes to the project file (i.e. implement your new conversion feature).
3. Commit changes in your local repo.
    a. Repeat step 3 from the previous section.
4. Push to the online repo.
    a. Note that, since a new branch has been created, Git is no longer sure where to push your commit. This is because your branch exists in your local repository but not in the online repository. As such, when pushing for the first time on a new branch, use the following command to create a corresponding branch in the online repo:
        i. `git push -u origin [branch name]`
5. Merge branch back into master.
    a. **In a terminal on the local project folder directory**, use the following commands:
        i. `git checkout master` (change working directory back to master)
        ii. `git fetch` (check online repository for most recent updates)
        iii. `git reset --hard origin/master` (reset master branch in local repo to match master branch in online repo)
            - The commands in (ii) and (iii) provide the same function as `git pull`. However, it is typically safer (i.e. less complex / more fool proof) to **reset** your local branch to match the online branch rather than to **merge** the online branch into your local branch. It is recommended that you use these commands instead of `git pull`.
        iv. `git merge origin/[branch name]` (merge branch into master)
        v. `git push` (reflect merge in online repo)
    b. Solve merge conflicts (if any) using the information below.

As you merge, you will find that your project has merge conflicts! Merge conflicts occur when developers working in parallel change the same lines of code and try to merge both of their updates together. As a result, Git isn't sure which lines to keep or which to scrap. In these scenarios, the developer currently merging is expected to review the code and solve the conflict. Keep in mind, developers tend to work on completely different features in a relatively large codebase. Typically, the chances that two people have changed the same lines of code in parallel is not very high. Outside of this assignment, merge conflicts are not something that happen all the time but they aren't uncommon.

Merge conflicts can be solved as follows:

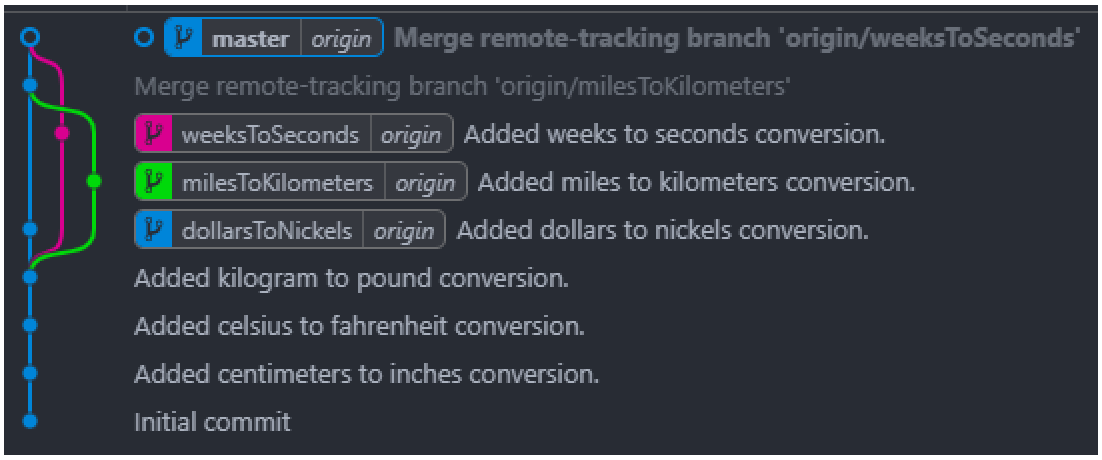
1. Open the file(s) containing merge conflicts (as indicated by your IDE or terminal).
    a. Once opened, for each conflict, you will see something similar to the below image. The changes labelled `HEAD (Current Change)` represent the code in the working branch (in this case, the master branch). The changes labelled `origin/[branch name] (Incoming Change)` represent the code being merged into the working branch. Note that the colored highlighting is purely visual. You are still able to edit/rearrange the code to your liking.


Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes | Start Live Share Session

```
<<<<<<< HEAD (Current Change)
    def dollarsToNickels(self, dollar_value):
        return dollar_value * 20
=======
    def milesToKilometers(self, miles_value):
        return miles_value * 1.60934
>>>>>>> origin/milesToKilometers (Incoming Change)
```

2. Examine the conflict and make appropriate changes.
    a. Examine both the Current and Incoming changes. In this case, it is clear that both pieces of code should be kept in the project. In other cases, you may also find that one of the changes is outdated and should be deleted or modified. For more complex merge conflicts, you may have to coordinate with the developer that your code conflicts with to ensure that your changes don't break their code.
3. Once all merge conflicts and resulting syntax errors have been solved in a particular file, save your changes.
4. Repeat the above for all files containing merge conflicts.
5. Once all merge conflicts have been solved in all files, stage the conflicted files.
    a. Use the following command:
        i. `git stage [file name with extension]` (i.e. Converter.py)
6. Once all the conflicted files have been staged, continue the merge. **Do not commit.**
    a. Use the following command:
        i. `git merge --continue`
7. Repeat the above if more conflicts are detected as the merge continues.
8. If there are no more conflicts detected, your IDE will open a file where you can edit your merge commit message. Typically, these don't need to be changed. Just save and close the file.
9. The merge should now be complete. Check your Git Graph to see the newly published merge commit.
10. If you are unsure about whether you solved the conflicts correctly, it may be a good idea to run the impacted code to see if it works as expected. If satisfied, push your merge commit to the online repo.

Once each group member has merged their parallel branches together, the resulting Git tree should look similar to the following image.


master | origin Merge remote-tracking branch 'origin/weeksToSeconds'
Merge remote-tracking branch 'origin/milesToKilometers'
weeksToSeconds | origin Added weeks to seconds conversion.
milesToKilometers | origin Added miles to kilometers conversion.
dollarsToNickels | origin Added dollars to nickels conversion.
Added kilogram to pound conversion.
Added celsius to fahrenheit conversion.
Added centimeters to inches conversion.
Initial commit

As you can see, 3 branches were created, a commit was published to each branch, and each branch was merged back into the master. Note that, since no updates had been committed to master before the dollarsToNickels branch was merged, the creation of a merge commit was not necessary and the dollarsToNickels branch just merged as an extension to the master branch. It's okay if your tree isn't exactly the same. What matters is that your master file now contains everyone's changes and remains functional.

Moving Forward

You should know be familiar with the basics of using Git. The initial learning curve is fairly steep. However, with practice, you'll become much more comfortable. If you are a software student, it is recommended that you use Git for all of your software assignments moving forward.

The processes of creating git projects, committing changes, and managing branches can be hard to remember initially. For future Git projects, a cheat sheet may be helpful. This tutorial only covered surface level git functionality. If you are interested in learning more, the next concepts to learn are listed in the Git Content section.