Contents

[Git 2](#_Toc99046374)

[Git command 2](#_Toc99046375)

[Screenshot Tips 4](#_Toc99046376)

[Getting Started 6](#_Toc99046377)

[After creating the account 6](#_Toc99046378)

[Creating A Repository 7](#_Toc99046379)

[Setting Up Of Initialized Repository 9](#_Toc99046380)

[Setting Up Of Empty Repository 12](#_Toc99046381)

[Enable Hidden File 17](#_Toc99046382)

[Repository 19](#_Toc99046383)

[Pulling Update From Remote Branch 21](#_Toc99046384)

[Understanding Of Changes And Conflicts 33](#_Toc99046385)

[Changes 33](#_Toc99046386)

[Conflicts 34](#_Toc99046387)

[Committing And Pushing Of Code 36](#_Toc99046388)

[Committing Of Changes 36](#_Toc99046389)

[Pushing To Remote Branch 41](#_Toc99046390)

[.gitignore file 43](#_Toc99046391)

# Git

Git is useful not just for code collaboration, it can act as a “google drive” with the power of going to different revision. This also means that you can get your entire repository(project) if you have access to your github account. Git is closely integration with visual studio code which allow you to better merge codes from other commits. Gitlens is an extension that is in visual studio marketplace is a useful extension when you work in a group because you can see who edited which part of the code and which commit.



**Local Branch/Repository** -> Branch/Repository that is on the local computer.

**Remote Branch/Repository** -> Branch/Repository that is on the github server.

# Git command

**git add <file\_name>** ->Add files/folders into staging and replace the <file\_name> with file name or period(.) to add all

**git branch** -> List all the branch that is in the repository

**git checkout <branch\_name>** -> Checkout to a branch in a repository and replace <branch\_name> with the branch name.

**git checkout -b <branch\_name>** -> To checkout a new local branch and replace <branch\_name> with the branch name.

**git clone <git\_repository\_url>** -> Clone(Download) the repository(project) to your local machine and replace <git\_repository\_url> with the git repository url that you can find it in your git repository.

**git commit -m ‘the message for commit’** -> Commit all the staging files/folders into local repository which are ready to push to remote branch, replace the message for commit with the message you want to label the commit with.

**git fetch** -> Fetch for new metadata from the repository, it can be fetching new branches that are push to the repository but not visible to you on your local device.

**git log** -> Show you all the commit of the current branch

**git push** -> Push the commits(changes) to the remote branch.

**git push -u origin <branch\_name>** -> Push the commits(changes) to the remote branch, replace <branch\_name> with the branch name of your local repository. Normally this command are used if you push to the branch for the first time.

**git pull** -> Pull changes from current remote branch to local. This will pull all the updates that is being pushed to the remote branch to your local branch.

**git pull origin <branch\_name>** -> Pull changes from a remote branch to local. This will pull all the updates that is being pushed to that remote branch to your local branch. Replace the <branch\_name> with the targeted branch name.

**git rebase origin/<base>** -> Let you put commits on top of the commit of a branch, change <base> to the branch name, [read more about rebase here](https://www.atlassian.com/git/tutorials/rewriting-history/git-rebase).

**git reset** -> Let you reset to the changes of the head. Use this command only if you understand what you are doing as different parameter might make all your changes to be gone forever. A hard reset would remove every changes that are uncommitted a soft reset will keep the changes that are uncommitted, [read more about reset here as it have lots of things to cover](https://devconnected.com/how-to-git-reset-to-head/).

**git status** -> Check files/folder that are currently untracked, modified and staged.

**git stash** -> Let you save your current changes without committing, [read more about stash here as it have lots of things to cover](https://devconnected.com/how-to-git-stash-changes/%23:~:text=Apply%20Git%20stashes,-Now%20that%20you&text=In%20order%20to%20apply%20your,the%20stack%20will%20be%20applied.%20).

# Screenshot Tips

Just a tip if you want screenshot in window machine you can press **win+shift+s** this will open snipping tool then left click and drag your mouse whichever part you want to screenshot after you release you should see a popup on your right side if not you can go to notification to see the screenshot.

Text, letter

Description automatically generated

Click this



You will find it there

Graphical user interface, text, application

Description automatically generated

Click on the screenshot which will open up this

Graphical user interface, application, Word

Description automatically generated

You can either copy the screenshot and paste on wherever you want or save it to your local machine.

# Getting Started

To get started, you must always have git on your machine which you can download [here](https://git-scm.com/downloads).

Create a github account [here](https://github.com/signup?ref_cta=Sign+up&ref_loc=header+logged+out&ref_page=%2F&source=header-home).

## After creating the account

Graphical user interface, application, Word

Description automatically generated

If you click top right logo you will be able to see your username and all other setting. The username can be used to add collaborator for private repository.

## Creating A Repository

If you are at github.com, you can create a repository with this button

Graphical user interface, text, application, chat or text message

Description automatically generated

If you are in any pages of github, you can click this button at the top right of the page and select new repository

Graphical user interface, text, application, chat or text message

Description automatically generated

You should see this

Graphical user interface, text, application, email

Description automatically generated

The repository name is the name of your project name.

Description you can fill in if you want but I always skipped that.

For Public and Private, it means that who can access if you chose Public means your repository is open to public anyone can access and modify the file. If you chose Private means that your repository can only be accessed only by you and any collaborator(if you have add) only.

For README file, this file is to tell people the instruction of how to run the project, this will be render in your github page. Mostly README file can always be found in those open-source repository.

For .gitignore file, as the name suggest it means that the file/folder you do not want to commit in for the repository. The period(.) infront of any file name/folder means that it is a hidden file(in ubuntu) which means you cannot see it in the file explorer but you can enable your window to see hidden file. For windows, a period(.) infront of the file/folder name will not hide unless you have changed it’s property to be hidden.

Choose a license, I have never used that before most of the time open-source library will state what kind of license it is. You can find all the license [here](https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/licensing-a-repository).

After you create the repository, you should see something like that if you have checked any of the checkbox during the creation. Continue with the following section [here](#_Setting_Up_Of). If you did not select any of the option, skip to [here](#_Setting_Up_Of_1) as the instruction is a little bit different.

Graphical user interface, text, application, email

Description automatically generated

## Setting Up Of Initialized Repository

First go to the directory that you want to clone(download) your repository, for this example I just going to clone it to my desktop. In the directory, right click and select git bash.

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

The yellow text is the current directory of the bash command prompt is currently in.

Next go back to your repository in github.

Press the code button which will show this popup

Graphical user interface, text, application, email

Description automatically generated

Press the copy button that is located beside the url.

Go back to the terminal(git bash).

To clone(download) your repository, use the command **git clone <repository\_url>** , if this is your first time git bash you have to configure it which they will prompt you to login your github account.

Text

Description automatically generated

Once you clone the repository you will see the file in the directory

A picture containing text, tree, outdoor

Description automatically generated

Open the folder.

Graphical user interface, application, email

Description automatically generated

When you open the folder, you should see the .git folder(ignore the other file for now). If you are unable to see the folder, you can enable it by reading [here](#_Enable_Hidden_File). This .git folder will help you keep track of all the changes you made. On a side note, the folder that contain .git folder is called the root folder. Anything file or folder you add inside here will be tracked even if it is a nested subfolder. You have successfully setup your repository till this step.

## Setting Up Of Empty Repository

Graphical user interface, text, application, email

Description automatically generated

You should see something like this.

If you do not have an existing folder you want to push to git you can create a empty folder. So open up the folder that you want to push to git.

Graphical user interface, application, Excel

Description automatically generated

Right click and select git bash here

Graphical user interface, application

Description automatically generated

Type in **git init** in the terminal(git bash) which will initialize an empty git repository.

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

You should see the .git folder if you don’t you can enable viewing of hidden file [here](#_Enable_Hidden_File).

Add in the files and folders that you want to push to the git repository in github.

After adding in your files/folders then type in **git status** in the terminal.

Text

Description automatically generated

You should be seeing the files/folders that are in the folder which is in red.

Next we have to add the files/folders into staging area which will let us commit and push to the remote repository. Type in **git add .** in the terminal, the period(.) means all the file that are untracked, you can add by file name too just replace the period(.) to the file name.

Once added, type in **git status**  in the terminal. You should see those file name turning green which means it have been staged.

Text

Description automatically generated

To this stage, we can refer back to the github.com instruction.

Graphical user interface, application

Description automatically generated

Copy paste starting from the line git commit to the last line, line by line. If you have not configure your git environment, you would be prompt to set your name and email through this 2 commands.

git config --global user.name "your\_name"

git config --global user.email your\_email

Text

Description automatically generated

You should see this after you execute all the command. Refresh the github.com page and you should see the repository updated.

Graphical user interface, text, application

Description automatically generated

## Enable Hidden File

Press the win key and search for hidden and press enter and this screen should show

Graphical user interface, application

Description automatically generated

Press show settings for Changes settings to show hidden and system files and you should see the following screen.

Graphical user interface, text, application

Description automatically generated

Select show hidden files, folders, and drives then press ok. Now you can see hidden files and folder. You can go back to the section you left off [Setting Up Of Initialized Repository](#_Setting_Up_Of_2) or [Setting Up Of Empty Repository](#_Setting_Up_Of_1)

# Repository

Graphical user interface, text, application, email

Description automatically generated

The top left is the repository owner username + repository name (in my case is chiayiquan / random\_test).

The tabs that are more important are the code, issues, pull request tab. Code tab will show you the branches, files in the repository, tags, number of commits and other stuff.

Graphical user interface, text, application, email

Description automatically generated

The issue tab let you create task and assign task to different collaborator in the repository. This is a good place to plan on feature and assignment of work when you are working in a team.

Graphical user interface, text, application

Description automatically generated

The pull request tab is for you to create a pull request to merge your code to the main/test branch. Usually this require approval from lead/senior developer and if there are test case, it will require the test to pass in order to merge to the desire branch.

# Pulling Update From Remote Branch

To pull update from the remote branch, you can do it either from the visual studio code or from the terminal.

Just a quick tip on how to open visual studio code from the file explorer.

In the folder you want to open the visual studio code, right click and select git bash here.

Graphical user interface, application

Description automatically generated

In the terminal type **code .** this will open the visual studio code of the current directory.

Text

Description automatically generated

To see git commits of the repository you can either use the git bash to check or use extension in visual studio code.

You would have to go to the extensions, search and install Git Graph.

A screenshot of a computer

Description automatically generated with medium confidence



At the bottom left you should see Git Graph click on it.

Graphical user interface, text

Description automatically generated



This will show up, there is three thing that you must know is that there is a blue ring that symbolizes the current commit your local branch is at. The main label means your local main branch current commit and the origin/main means that the remote main branch current commit.

Graphical user interface, text

Description automatically generated

From the screenshot above, my current local main branch is 3 commits behind the remote main branch. To pull the updates through visual studio code, select the third button(Source Control) on the sidebar of visual studio code then hover to the text of the Source Control shown in the screenshot below and press the three dots. Inside there is a lot of option so just press pull for now.

Text

Description automatically generated



Once pulled, the current local branch will be updated to be the same as the remote branch.

Text

Description automatically generated

To update the local repository with terminal, you can do it either from git bash or do it from visual studio code terminal both are the same. In your visual studio code you can press **ctrl+`** and the terminal should pop up at the bottom. You should see it in bash if it isn’t in bash, press the expand icon beside the plus(+) icon then select git bash.

Shape, rectangle

Description automatically generated



A picture containing text

Description automatically generated



In the terminal, type in **git pull** and it will update the content to the latest commit.

Text

Description automatically generated with medium confidence

When you have changes, and you want to pull the remote repository, you have 2 options. One is using rebase and the other method is using stash. For this example, my local branch current commit is at f8a6b74 and you can see that my changes derailed from the remote branch.

A screenshot of a computer

Description automatically generated with medium confidence

If I do a git pull, I will get an error and not able to do that.

Text

Description automatically generated

The first method is to use the stash method.

Using the visual studio code, click the third button at the which will show you the Source Control. Hover your mouse to the Source Control and you will see all the option press the three dots(…), hover to stash and choose stash (include Untracked).

Graphical user interface

Description automatically generated



You can do it in the terminal too. By typing **git stash -u** or **git stash --all**

Text

Description automatically generated with medium confidence

After stashing your changes, you can do a pull for the latest update.

Text

Description automatically generated

So now we can pop our stash. Using the visual studio code, click the third button at the which will show you the Source Control. Hover your mouse to the Source Control and you will see all the option press the three dots(…), hover to stash and choose Pop Latest Stash

Graphical user interface, application

Description automatically generated



Or in the terminal you can type **git stash pop** to pop the latest stash.

Text

Description automatically generated

So after popping the stash I have conflict so I have to fix it before I can commit.

A screenshot of a computer

Description automatically generated

So after fixing the conflict, I can stage the changes by press the plus(+) button or type in **git add .** in the terminal.

Text

Description automatically generated

So once added I will just commit these changes. So after committing it, the commit will be on top of the remote branch latest commit and from here you can push all the changes to the remote branch.

Text

Description automatically generated

So the other method is to use rebase, what this mean is you have to the commit the files first then place the commit above the latest commit with rebase command. I have committed the changes as you can see my local branch commit is above the remote branch commit however it is not in the same “track” as the remote branch, if a force push is done here the “Update thirdCommitFunction” will replace all the pink track commit which is not what we want.

Graphical user interface, text, application

Description automatically generated

If you do through visual studio code, click the third button at the which will show you the Source Control. Hover your mouse to the Source Control and you will see all the option press the three dots(…), hover to Pull,Push then select Pull(Rebase).

A screenshot of a computer

Description automatically generated with medium confidence



Once you pull and rebase you will see the “track” flip the local branch commit will be in the right hand side. In my case because there is conflict I have to resolve it first, stage it and commit.

Graphical user interface, text, application

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

As you can see the commit now is above the remote branch commit.

Graphical user interface, text, application

Description automatically generated

The other way is to use the terminal to run the rebase command.

So open up the terminal by pressing **ctrl+`** and type in **git pull --rebase** , **git pull origin <branch\_name> --rebase** if you are pulling from another branch. You can use **git rebase origin/<branch\_name>** too is the same exact command.

A screenshot of a computer

Description automatically generated with medium confidence

If there is no conflict between the commit you pulled it will act like a normal git pull but if there is conflict between the commit you pulled, you will have to fix the conflict and add it to staged changes. Just a few things to notice here during rebasing you can see the commit id with a tag (Rebasing) at the bottom and the blue text in the terminal will show branch\_name/REBASE 1/1. If you want to cancel the rebase type **git rebase --abort** in the terminal it will cancel the rebase. If you want to skip the current commit you can type **git rebase --skip** and to continue the rebase just have to type in **git rebase –continue**

Text

Description automatically generated



So once I run the **git rebase --continue** in the terminal, the commit will be on top of the latest commit in the remote branch.

Text

Description automatically generated

# Understanding Of Changes And Conflicts

## Changes

Changes are the content of the code/file you changed locally, in visual studio code, you can see the local code changes by pressing the third button on the side bar(source control)

A screenshot of a computer

Description automatically generated with medium confidence

If you click on the file in this tab, you will see what you have changed so far. This will opened up the file, the left side is the current content on the latest remote branch, the right is the changes you did locally. Unfortunately, this only work on codes, it won’t work on file like word, pdf etc… but any changes will also be tracked.

A screenshot of a video game

Description automatically generated with medium confidence

## Conflicts

Conflicts will always occur when you work in a team or you work on different pc without always pulling the latest commit. When your teammate pushed their code and the changes they made are the same file as yours, there will be a conflict when you pull. For example, the secondCommitFunction was committed and pushed to the remote branch first but then I didn’t pull the update so I edited the same file with a new function called thirdCommitFunction. So when I pulled the changes, I will have conflict showing that there is changed content of the same file. The green part which mean the changes that are on the remote branch and the bottom blue part are changes you made.

A screenshot of a computer

Description automatically generated with medium confidence

So I have 3 option is either accept current change which is the remote branch change it will remove the blue part

Text

Description automatically generated

If you accept incoming changes meaning your changes, the green part will be removed

Text

Description automatically generated

If you select accept both changes it will keep both changes.

Text

Description automatically generated

So when you work in team, you have to be careful as your changes that override their might cause their code to stop working so always test the code after the merge then push out the code.

# Committing And Pushing Of Code

## Committing Of Changes

Commit of changes means that you “wrap” all your changes and add tie it to a commit id with it’s message. By clicking the third button(source control) in visual studio code you should see all the changes you made so far.

A screenshot of a computer

Description automatically generated with medium confidence

For this example I shifted the secondCommitFunction to the bottom of the file, you can see the left side means I removed the two line from the original copy which is highlighted in red and the new changes is the one that is on the right and is highlighted in yellow. So in order to commit this file, I will have to stage it. There is two option on how to stage the file, 1 is from the visual studio code which is shown in the screen shot, you can add file 1 by 1 all add all together.

A screenshot of a computer

Description automatically generated with medium confidence



I will go through all the icon shown in the screenshot briefly. The first icon is basically opening the file, the second icon means to discard all the change. The plus(+) icon means add to staging, so when clicked you should see that file going into a section called staged changes. The M tag means modified means there’s changes, the U tag means untracked which means git is not keep track of the file, the D tag means deleted. You can make the icon to appear just by hovering to the file.

Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, text, application

Description automatically generated



You can click the plus(+) icon in Changes to stage all the file and all will appear in the Staged Changes.

Graphical user interface, text, application

Description automatically generated

The minus(-) icon in Staged Changes means unstaging of the file from Staged Changes.

A screenshot of a computer screen

Description automatically generated with medium confidence



So once the minus icon is clicked it will go back to changes.

A screenshot of a computer

Description automatically generated with medium confidence

So once you have added all the file you can commit the changes by typing the message in this textbox. The message can be anything that is meaningful to the changes, for my habit I always use Add/Update/Delete the files/function.

Graphical user interface, text, application

Description automatically generated

After inputting the message, you can either press **Ctrl+Enter** or press the tick on the top. Once you committed, the Staged Changes section should disappear.

Graphical user interface, application

Description automatically generated



You have successfully committed locally, to see the commit log, just click on the Git Graph at the bottom left of your visual studio code.

Graphical user interface, text

Description automatically generated



From here you can see that my local main branch is 1 commit ahead of the remote main branch so when I push the commit it will be at the same level again. I will show again after I push this commit.

Graphical user interface, text, application

Description automatically generated

The other way of committing is through terminal/git bash. In your visual studio code you can press **ctrl+`** and the terminal should pop up at the bottom. You should see it in bash if it isn’t in bash, press the expand icon beside the plus(+) icon then select git bash.

Shape, rectangle

Description automatically generated



A picture containing text

Description automatically generated



So inside the terminal type **git status** You should see the file that you changed or the file you added.

Text

Description automatically generated

So inside the terminal type **git add filename**(you can actually type the initial letter then press tab to auto fill)if you want to add all the files, type in **git add .**

Text

Description automatically generated

To commit file in terminal type in **git commit -m ‘the\_message’**

Text

Description automatically generated

To see whether you have committed you can either use the extension Git Graph or type in terminal **git log --oneline** you will see the following

Text

Description automatically generated

## Pushing To Remote Branch

To push the commit to the remote branch you can use the visual studio code or use terminal to push the commit(s). You don’t have to commit 1 time and push to the remote branch you can stack up multiple commits to push to the remote branch. In order to push to the remote branch make sure that your commit is above the commit in the remote branch if not you will not be able to push to remote branch.

You can use the visual studio code to push by just hovering onto the source control then you should see the three dot press on it you should see all the option.

Text

Description automatically generated



Press the push to push to the remote branch

Text

Description automatically generated

To push to remote branch with terminal, just type in **git push** then it will be pushed, however if it is a branch that is not on the remote repository, you must do **git push --set-upstream origin <branch\_name>**

Text

Description automatically generated

Text

Description automatically generated

After the push, the origin/main and your local main will be at the same level

Text

Description automatically generated

Background pattern

Description automatically generated

# .gitignore file

Some of the times we do not want to push modules/build/env file to the github because it will take a very long time to clone the repository so we can put it in this .gitignore file which will stop tracking all this files when you make changes. The .gitignore must be in the root directory of the folder so right click in the directory and select **Git Bash Here** and in the terminal type **touch .gitignore** and it will create a .gitignore file.

Text

Description automatically generated

So open up your .gitignore file in a notepad or some editor. The # is comment, Builds/\* means that ignore all the content that is in Builds folder and !Builds/include is to exclude the ignore of the content in include folder. The **!** means exclude it from the ignore list.

Graphical user interface, application

Description automatically generated

This is what is inside my Builds folder.

Graphical user interface, application

Description automatically generated

After the .gitignore have a list of content of what to ignore and what to exclude from the ignore you can just keep using **git add .** it won’t add those ignored file into tracking.

Text

Description automatically generated

That’s all for the guide if you need reference to the github repository you can clone from [here](https://github.com/chiayiquan/git_guide.git).