12 Days of Git Combined

- WHAT IS VERSION CONTROL
- WHAT IS GIT
- INSTALLING GIT (COMMAND LINE AND GUI OPTIONS)
- BASIC GIT COMMANDS TO GET STARTED
- INSPECTING A REPOSITORY
- FILE OPERATIONS
- UNDOING COMMITS & CHANGES
- REWRITING HISTORY
- BRANCHES
- MERGING
- REBASING
- STASHING
- CONFLICTS
- SUBTREES AND SUBMODULES

SOURCE: https://github.com/weeyin83/14daysofgit

@arjun-panwar

12 Days of Git

DAY 1

@arjun-panwar

What is version control?

Every change to the code is recorded by version control in a form of database.

If a mistake is made, you can go back in time, compare it to prior versions, and help in fixing the error while causing the least amount of interruption to other people who are working on that code.

Version control isn't just for software developers.

For more visit: https://www.techielass.com/what-is-version-control/

What is Git?

Git is a tool that is used for code management. It is a free and open source tool.
Git is a tool that can be used within your DevOps workflows to maintain good version control.
Linus Torvalds created Git in 2005.

Version control is the theory, the process that is used to do that. Git is the tool that allows you to put that theory into action.

For more visit: https://www.techielass.com/what-is-git/

What is Git?

Refer this website to install git!
https://www.techielass.com/installing-git/

Configure Git for use

Before we start to use Git there are a few things that we need to configure. Setting your name and your email address is the first step. This information is use to record any changes you make to Git repositories.

```
git config --global user.name "Your Name"
git config --global user.email “you@example.com”
```

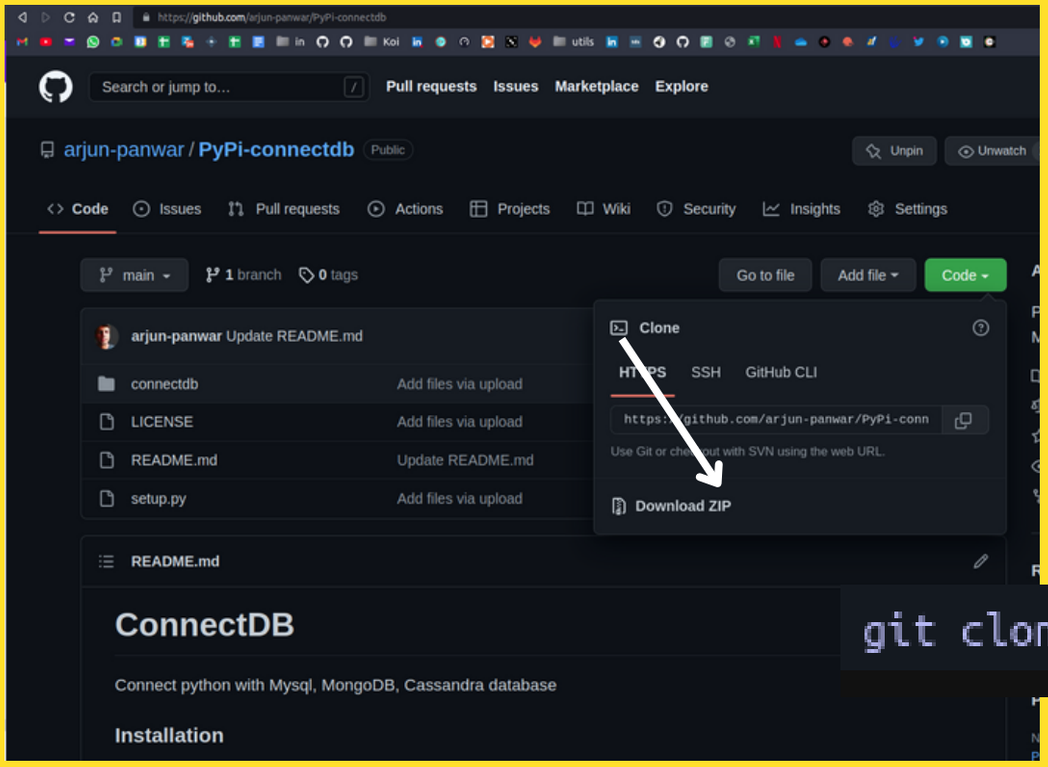
12 Days of Git

DAY 2
BASIC GIT COMMANDS

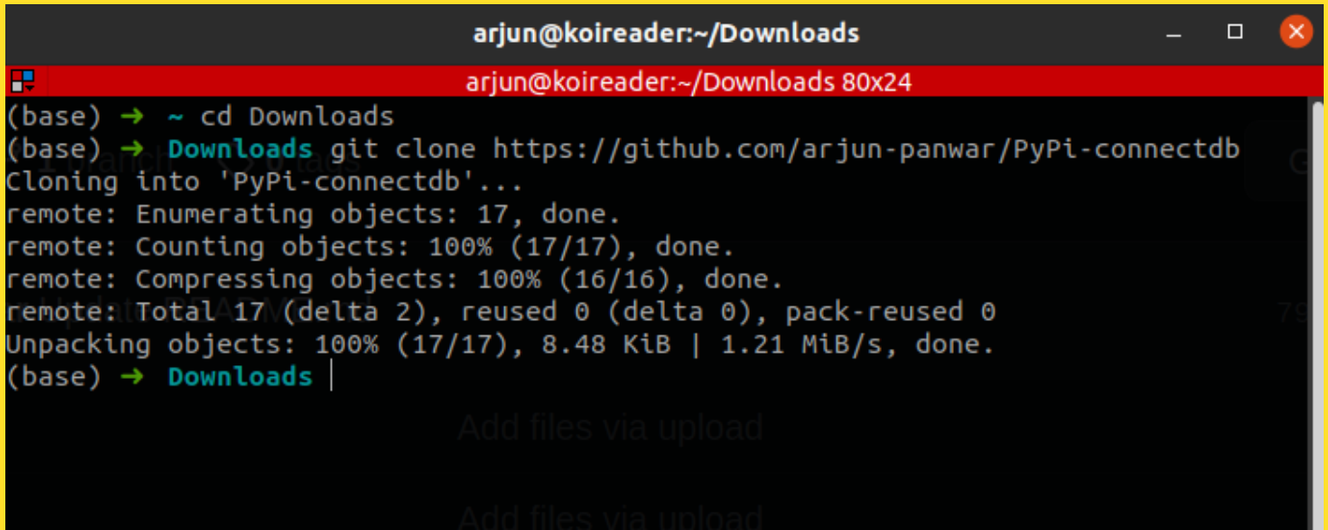
@arjun-panwar

Git Clone

Git clone is the command that will help you download an existing remote repository (like GitHub for example) to your machine.
For example if you head over to any GitHub repository and click on the green Code button. It will display a URL. Take a copy of that URL.




```
git clone https://repository-url.link
```



Git add

When we work on a local copy of a repository we will be creating, modifying, or delete files. When we do that we need to add those changes to our next commit so they are copied to the remote repository.
Git add starts to prepare our changes ready to be saved and sent to the remote repository.

We can add a single file with the command:

```
git add filename
```

Or we can add all the files and changes we made with the command:

```
git add -A
```

Git commit

There comes a point when you are working on your local repository that you want to save your changes.

Git commit is like setting a checkpoint. You can come back to this checkpoint at a later point if you need to.

When we issue the git commit command we need to issue a message to explain the changes we have made. This will help you or the next person along understand what has changed.

```
git commit -m "I have made changes to the code to add a new feature"
```

Git commit

There comes a point when you are working on your local repository that you want to save your changes.

Git commit is like setting a checkpoint. You can come back to this checkpoint at a later point if you need to.

When we issue the git commit command we need to issue a message to explain the changes we have made. This will help you or the next person along understand what has changed.

```
git commit -m "I have made changes to the code to add a new feature"
```

Git push

Once we have committed our changes we need to send them to the remote repository. We can do this using the push command.
The git push command uploads the commits you have made to the remote repository. Only committed changes will be uploaded.

```
git push
```

If you wanna push to a particular branch on origin, use:
git push origin <branch-name>

Git pull

Git pull is the command you would use to get any updates from the remote repository.
This command actually does two things in the background. It is a combination of two other commands, git fetch and git merge

- First of it it gets the updates from the remote repository (git fetch)
- Applies the changes to your local copy of the repository (git merge

```
git pull
```

There are times when you will use this command it will throw up errors or conflicts that you need to resolve but it will pull down any changes for you.

Git init

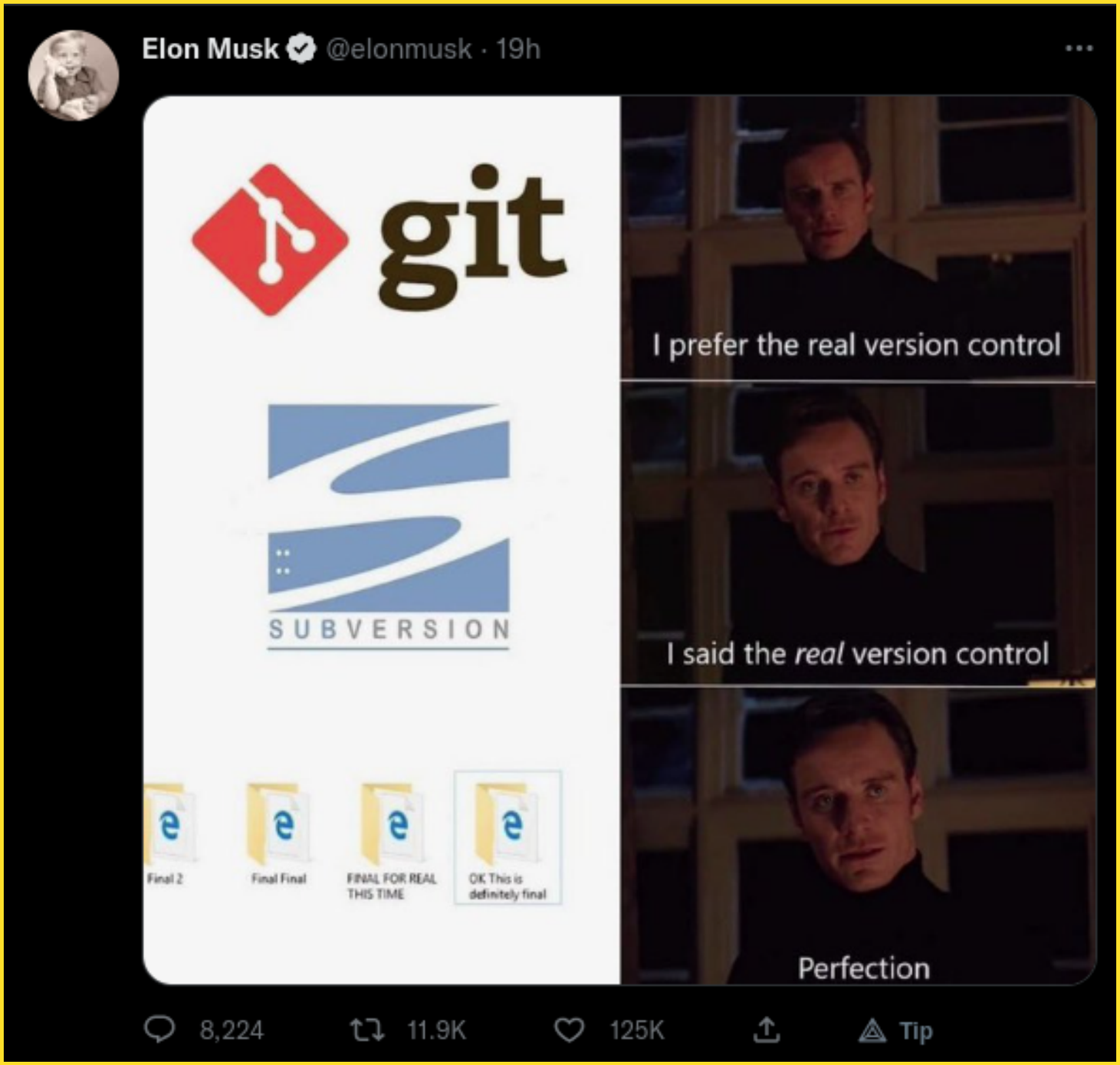
Git init can convert an existing project or folder into a Git repository.
Most git commands are not available until a repository is initialised, so typically this is the first command you will run when starting a new project on your local machine.
When you use git init it create a .git subdirectory in the working directory. This contains all the necessary Git metadata for the new repository.
You can use git init in a couple of way.
If you have an existing folder structure with files and code within it that you want to turn it into a git repository you can issue the command:

```
git init
```

If you haven't created a folder or started creating any code you can issue the following command to create the folder and initialise it as a git repository:

```
git init NewProject
```

For more visit:
https://www.techielass.com/basic-git-commands-to-get-started/



12 Days of Git

DAY3

INSPECT A GIT REPOSITORY

@arjun-panwar

There are times when you will want to check the status of a repository on your local machine, or when you want to look back at the history.

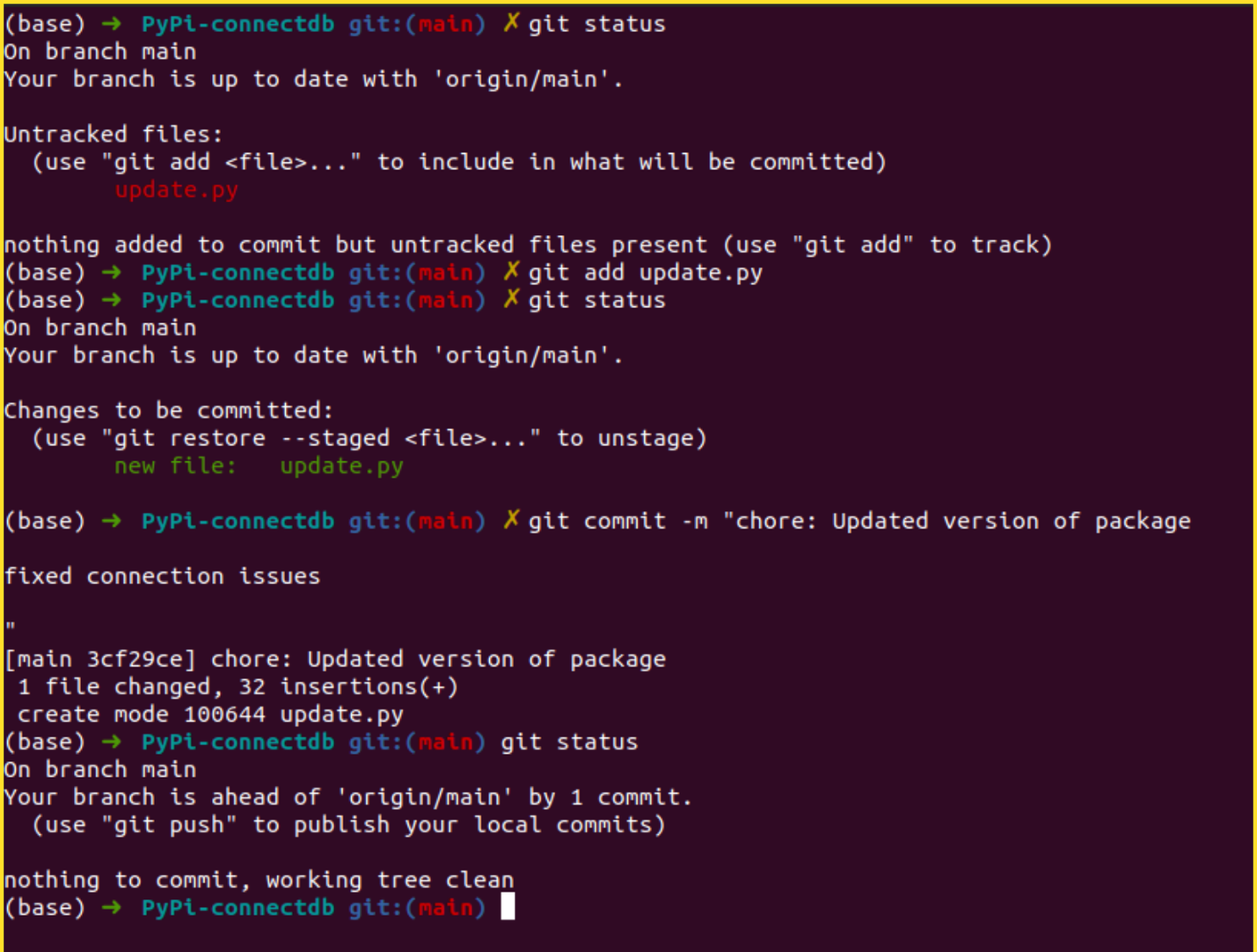
Maybe you haven't touched the repository for a few days and want to check what state you left it in.

Or maybe you have an issue and want to understand why and how to resolve it.

Git status

Git status displays the state of the working directory and the staging area. It will show you any changes which have been staged, which haven't and any files that Git is not tracking.

Git status won't show you the commit history of the repository though. It also shows that changes on the local repository that should be pushed to the remote repository.



Git log

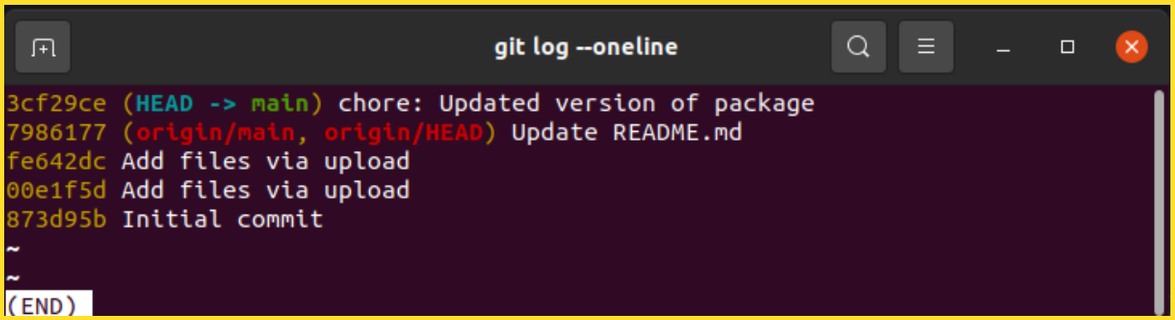
Git log is the command you want to use to see the repositories commit history.

The **space bar** to help you **scroll down** and you can **exit** it using **q**.

```
git log
```

```
commit 3cf29cede155e5378b1540150f04f659f3f2f8c2 (HEAD -> main)
Author: Arjun Panwar <arjun.panwar@koireder.com>
Date:   Tue Nov 8 22:37:45 2022 +0530

    chore: Updated version of package

    fixed connection issues

commit 79861774144aaff2c0b543bfbf2da04e8b7fcf25 (origin/main, origin/HEAD)
Author: Arjun Panwar <59289088+arjun-panwar@users.noreply.github.com>
Date:   Fri Sep 3 01:05:13 2021 +0530

    Update README.md
```

```
git log --oneline
```



To display the last three commits for you.

```
git log -n 3
```

To see what files were altered and how many lines were added or removed for each commit

```
git log --stat
```

Git blame

Git blame is a useful command if you want to understand who authored a certain change in the history of the repository. You can't just use git blame across the repository you need to specify a specific file you want to query.

```
git blame README.md
```

```
fe642dcb (Arjun Panwar 2021-09-02 14:56:30 +0530   1)
fe642dcb (Arjun Panwar 2021-09-02 14:56:30 +0530   2) # ConnectDB
fe642dcb (Arjun Panwar 2021-09-02 14:56:30 +0530   3)
fe642dcb (Arjun Panwar 2021-09-02 14:56:30 +0530   4) Connect python with Mysql, MongoDB, Cassandra database
fe642dcb (Arjun Panwar 2021-09-02 14:56:30 +0530   5)
fe642dcb (Arjun Panwar 2021-09-02 14:56:30 +0530   6) ### Installation
79861774 (Arjun Panwar 2021-09-03 01:05:13 +0530   7) Run below commandto install whole package
fe642dcb (Arjun Panwar 2021-09-02 14:56:30 +0530   8)
fe642dcb (Arjun Panwar 2021-09-02 14:56:30 +0530   9) ```pip install connectdb```
fe642dcb (Arjun Panwar 2021-09-02 14:56:30 +0530  10)
79861774 (Arjun Panwar 2021-09-03 01:05:13 +0530  11) For only Mysql
79861774 (Arjun Panwar 2021-09-03 01:05:13 +0530  12) ```pip install -U connectdb[mysql_db]```
79861774 (Arjun Panwar 2021-09-03 01:05:13 +0530  13)
79861774 (Arjun Panwar 2021-09-03 01:05:13 +0530  14) For only MongoDB
79861774 (Arjun Panwar 2021-09-03 01:05:13 +0530  15) ```pip install -U connectdb[mongodb]```
79861774 (Arjun Panwar 2021-09-03 01:05:13 +0530  16)
79861774 (Arjun Panwar 2021-09-03 01:05:13 +0530  17) For only Cassandra
79861774 (Arjun Panwar 2021-09-03 01:05:13 +0530  18) ```pip install -U connectdb[cassandra_db]```
79861774 (Arjun Panwar 2021-09-03 01:05:13 +0530  19)
fe642dcb (Arjun Panwar 2021-09-02 14:56:30 +0530  20) ## Mysql
fe642dcb (Arjun Panwar 2021-09-02 14:56:30 +0530  21)
fe642dcb (Arjun Panwar 2021-09-02 14:56:30 +0530  22) #### prerequisite
fe642dcb (Arjun Panwar 2021-09-02 14:56:30 +0530  23) mysql-connector-pythonmust be installed
fe642dcb (Arjun Panwar 2021-09-02 14:56:30 +0530  24) To install it, run below command
fe642dcb (Arjun Panwar 2021-09-02 14:56:30 +0530  25)
fe642dcb (Arjun Panwar 2021-09-02 14:56:30 +0530  26) ```pip install mysql-connector-python```
fe642dcb (Arjun Panwar 2021-09-02 14:56:30 +0530  27)
```

For more visit:
https://www.techielass.com/inspect-a-git-respository/

12 Days of Git

DAY 4
FILE OPERATIONS WITH GIT

@arjun-panwar

Rename a file

Git mv is the command that can help you with renaming. We do need to understand what this command does though.

Git mv is the equivalent to the following three commands:

- mv old_file.md new_file.md
- git add new_file.md
- git rm old_file.md

The mv command is a Unix/Linux command that is used to change the file name. The git add command is used to stage the new version of the file. The last part git rm removes the old file from being tracked.

Lets see **git mv** in action

git mv <old file path> <renamed file path>

```
git mv update.py update_connection.py
```

```
(base) ➜  PyPi-connectdb git:(main) ✗ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:    update.py -> update_connection.py
```

Moving a file

Git mv is also the command I can use to move a file from one location to another within my repository.

git mv <old file path> <new file path>

```
git mv ./update_connection.py ./connection/update_connection.py
```

```
(base) → PyPi-connectdb git:(main) ✗ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:    update.py -> connection/update_connection.py
```

Git mv options

Git mv -f

With the -f option we can tell Git we are okay to overwriting the destination with our new file. It's basically forcing any renaming or move you want to happen. Be cautious with this as you could overwrite something you need.

Git mv -k

The -k option allows Git to skip over any erroneous conditions resulting from a git mv call. For example, if you are trying to move a file to another location and that file already exists the command would error out. If you don't want to see that error and have Git move onto your next instruction you should use the -k option.

Git mv -n

The -n option is actually short for --dry-run. It won't actually carry out the move or rename, it will just show you what would happen if you did perform the command.

Git mv -v

The last option is the verbose option. Using this option will give you more information and feedback when you execute the command.

For more visit:
https://git-scm.com/docs/git-mv

12 Days of Git

DAY 5
UNDOING COMMITS & CHANGES

@arjun-panwar

Undoing commits & changes

There are times where you will be made a commit and realised that you want to undo that for whatever reason.

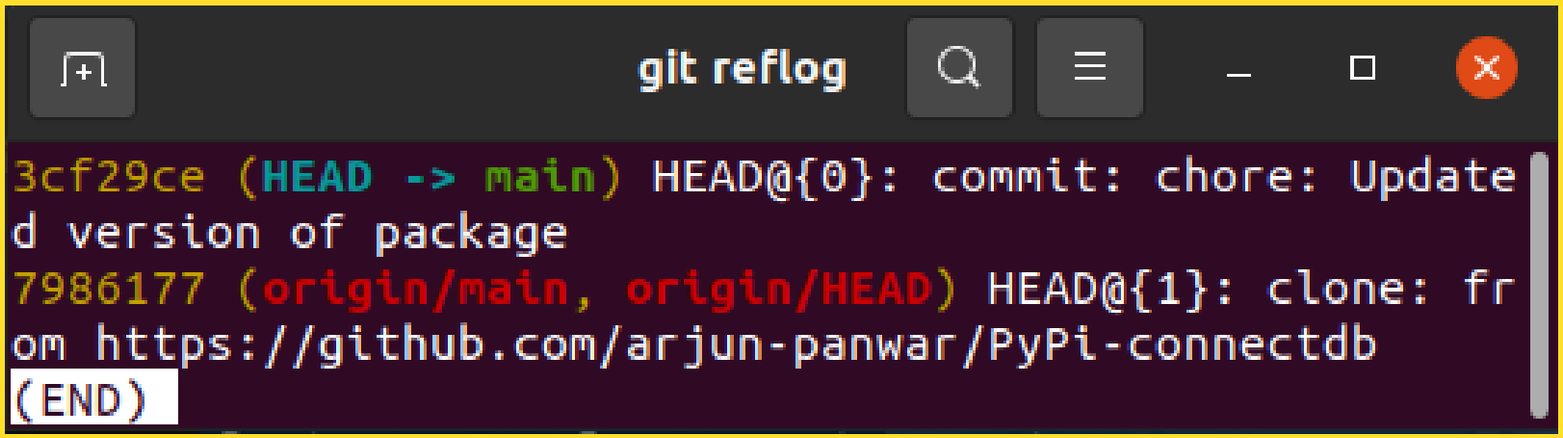
What's the best of doing that though? Deleting the file, you just created, deleting the line you wrote in a file? What happens when you've made a ton of changes and don't remember all the bits that need undone now?

This is where git revert, and git reset can help.

Git Revert

Git revert is a command that can remove all the changes a single commit made to your repository.

Used **git reflog** to get more details about comit



From **git reflog** we can get commit number

So if we wanna revert commit **3cf29ce**

```
git revert 3cf29ce
```

It's important to remember with this command, you are only reverting the commit you aren't erasing the history of the commit. So, any changes can still be referenced within the history of the repository.

Git Reset

With Git revert we just undid changes from a specific commit. However, there might be occasions where you want to revert every change that has happened since a given commit.
This is where the git reset command can be used.

The steps to use Git reset are:

1. Use **git reflog** to get the commit number you wish to reset to
2. Issue the command **git reset number**
3. The repository will not reset your repository to the state it was at that choosen commit

```
(base) ➜  PyPi-connectdb git:(main) ✗ git reset 3cf29ce
Unstaged changes after reset:
D       update.py
```

Again, with the git reset command remember that you are just reverting to a previous state, you aren't removing the history. It will still be there to see and refer to.

For more visit:
https://www.techielass.com/undoing-commits-changes/

12 Days of Git

DAY 6
REWRITING GIT HISTORY

@arjun-panwar

Rewriting History

Change most recent Git commit message

There are times when you make a commit but realise you've written the wrong thing within the commit message. What do you do?

This is one of the use cases for the --amend option.

```
(base) ➜  PyPi-connectdb git:(main) ✗ git commit --amend -m "updated last commit massage"
[main 968cb72] updated last commit massage
 Author: Arjun Panwar <arjun.panwar@
 Date: Tue Nov 8 22:37:45 2022 +0530
 1 file changed, 32 insertions(+)
 create mode 100644 update.py
```

```
commit 968cb72afde456eac5c45cbc2611d4643e47101b (HEAD -> main)
Author: Arjun Panwar <arjun.panwar@
Date:   Tue Nov 8 22:37:45 2022 +0530

    updated last commit massage
```

It will replace last message

Add extra changes to a commit

There are times when you complete a commit and then realise you want to add in one more change or you've forgotten something. And it would make more sense to add it into that commit rather than open another one.

This is another use case for the **git commit --amend** command.

Your workflow might be:

1. Make changes to file 1 and file 2
2. Add and commit those changes
3. Realise you've forgotten to add a small change into file 1
4. Make the additional change
5. Use the command ***git commit --amend --no-edit***

The additional option on the command, **--no-edit** takes that last change and puts it into the previous commit, without changing the message. For anyone else looking at this commit, it will look like it was done in one commit.

I would caution using this option and only using it on your own commits, don't confuse others by amending other people's commits

For more visit:
https://www.techielass.com/rewriting-git-history/

Other ways to rewriting history will be explored in comming days

12 Days of Git

DAY 7

GIT BRANCHES

@arjun-panwar

What are Git branches?

Branches are an everyday part of the process when using Git. Effectively they are a point to your changes. You might create a new branch when you want to work on a new feature, or bug fix and keep those changes completely separate until you are ready to release them.

There are a few commands that you can use to work with branches, let's take a look at them.

Create a new branch

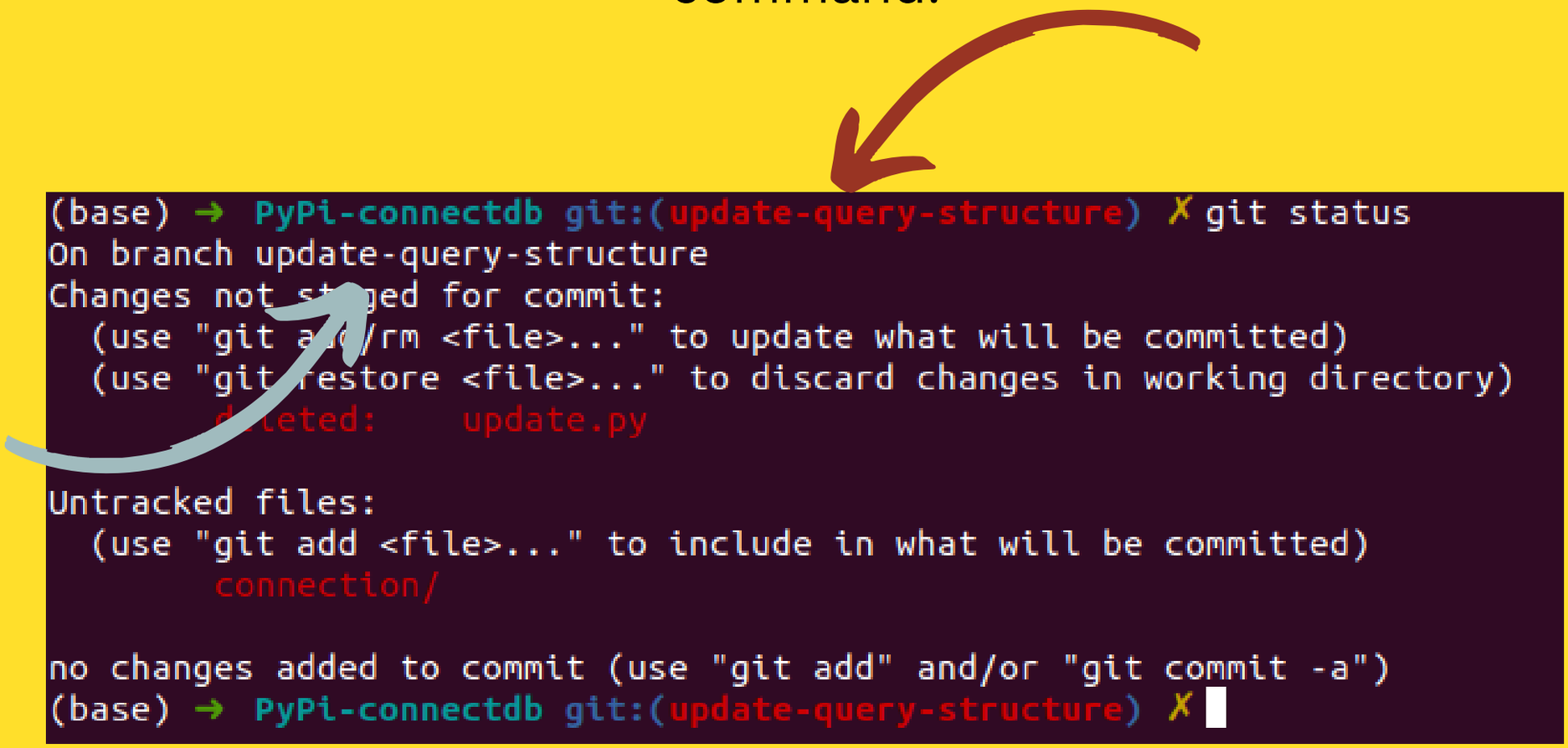
If you have a copy of a repository and you want to make changes then creating a new branch is best practice.

To create a new branch you can run the command:

```
(base) ➜  PyPi-connectdb git:(main) ✗ git checkout -b update-query-structure
Switched to a new branch 'update-query-structure'
(base) ➜  PyPi-connectdb git:(update-query-structure) ✗
```

See what branch you are on

When you are working within a repository, and you want to check what branch you are on you can use the following command:



Its also visible in terminal base line

Switch to a local branch

If you have a branch within your local repository that you've created then you can easily switch to it using the following command:

```
(base) ➜ PyPi-connectdb git:(update-query-structure) ✗ git checkout main
D       update.py
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)
(base) ➜ PyPi-connectdb git:(main) ✗
```

Switch to a branch that's came from a remote repo

If you have cloned a Git repository from a remote location and it's come with a bunch of branches, then you can use the following command:

```
git checkout --track origin/main
```

Push a branch

When you create a branch on your local copy of the repository it won't automatically create within the remote location. When you come to push your changes to that remote location you can't just use the git push command you need to use a slightly different command, either of these:

Using **branch name**

```
git push -u origin branchname
```

Using **HEAD**

Referencing HEAD saves you from having to type out the exact name of the branch.

```
git push -u origin HEAD
```

Of course, if your branch already exists in the remote location, you can just run git push.

Merge a branch

So, you've created a branch, and you are ready to merge that branch into the main one, ready for production or the next step in your development phrase. How can you do it?
The first thing you need to do is switch to the branch you want to merge your changes into. In this example I want to merge my changes from the branch "update-query-structure" into my main branch.

```
(base) → PyPi-connectdb git:(update-query-structure) ✗ git checkout main
D       update.py
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)
(base) → PyPi-connectdb git:(main) ✗ git merge update-query-structure
```

When you issue this command you may receive merge conflicts, we will be looking at how to deal with them in coming days.

I now have to push this merge from my local repository to my remote repository, I do that with the command:

```
PyPi-connectdb git:(main) ✗ git push
```

Delete a local branch

If you want to delete a branch that has already been merged you can use the command:

git branch -d branchname

If you want to delete a local branch regardless of whether it has been merged or not, then the command to use is:

git branch -D branchname

There is a subtle difference, the capital **D** is really a shortcut for **--delete --force.**

For more visit:
https://www.techielass.com/git-branches/

12 Days of Git

DAY 8
MERGING WITH GIT

@arjun-panwar

What is merging?

Merging is where you take two branches and combine them. Git will take the commit pointers it has within the two branches and attempt to find a common base commit between them and then unify the two branches.

Merge two Git branches together

You've got a branch and it's ready to merge into your main one. The first thing you need to do is make sure you are in the branch you want to merge your changes into.
In this example I have a branch called **"update-query-structure "**, and I want to merge it with my main branch. I first switch over to my main branch:

```
(base) ➜  PyPi-connectdb git:(update-query-structure) ✗ git checkout main
D       update.py
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)
(base) ➜  PyPi-connectdb git:(main) ✗
```

Now I am in the branch I want to merge my changes into I type:

```
git:(main) ✗ git merge update-query-structure
```

I now have to push this merge from my local repository to my remote repository, I do that with the command:

```
PyPi-connectdb git:(main) ✗ git push
```

Merge more than one branch

You can merge more than one branch at a time. There could be conflicts and issues when doing so, and we'll be looking in future.

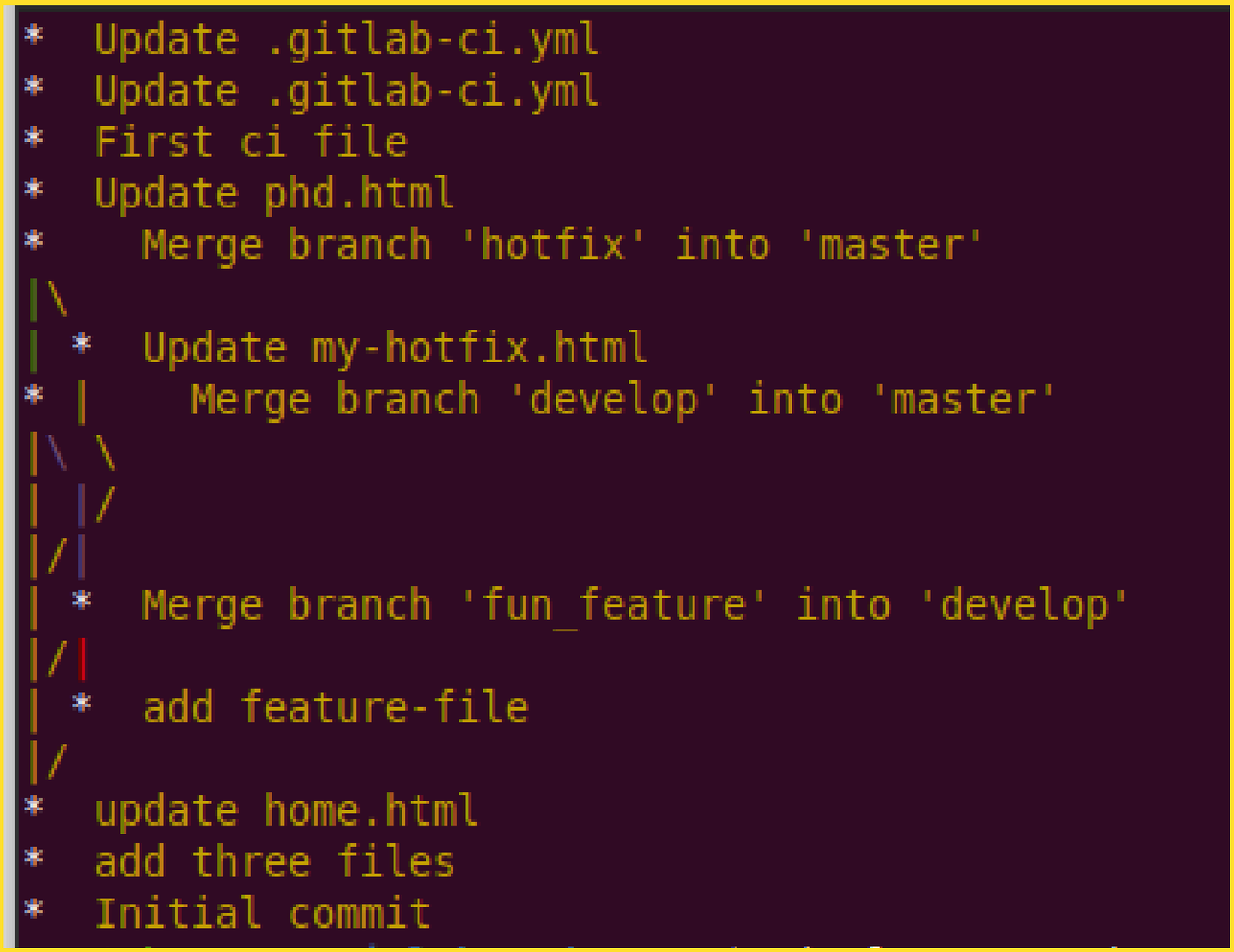
In this example, I have 2 branches with minor changes, and I want to merge them into my dev branch.
I issue the command
git merge update-query-structure main

```
git merge update-query-structure main
```

View the log of merging

We saw that there is a command called git log that we can use to find out more information about the current state of the repository. If we append more options to that command, we can get a graphical version of our commits.

```
git log --graph --oneline
```



Git merge strategies

When digging into merging, I found there were different merge strategies available for different use cases.

Merging with Git - 14 days of Git

Look at how you can use Git commands to merge your commits.

Techielass / Sep 29

Learn more about them on

https://git-scm.com/docs/merge-strategies

For more visit:
https://www.techielass.com/merging-with-git/

12 Days of Git

DAY 9

GIT REBASING

@arjun-panwar

What is Git Rebasing?

Rebasing your Git repository rewrites its history. **It can be a harmful command**, so it is one to watch when you use it. Typically, you would use the git rebase command for one of the following reasons:

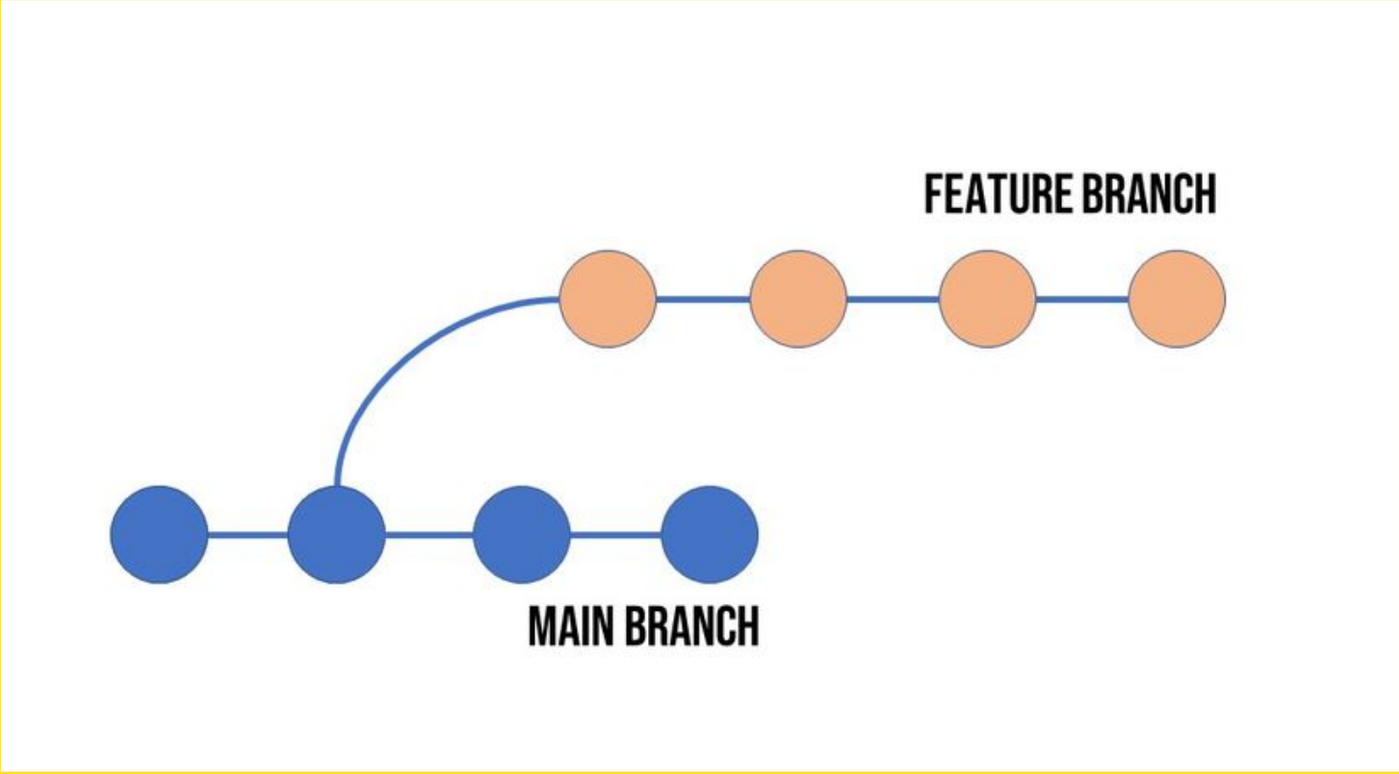
- Edit a previous commit message
- Combine multiple commits into one
- Delete or revert commits that are no longer necessary

Why do we need Git rebase?

Rebasing rewrites the project's history. It gives you a much cleaner project history. Rebasing eliminates the unnecessary merge commits required by git merge. It gives a much linear history when you are looking back at your logs.

Why do we need Git rebase?

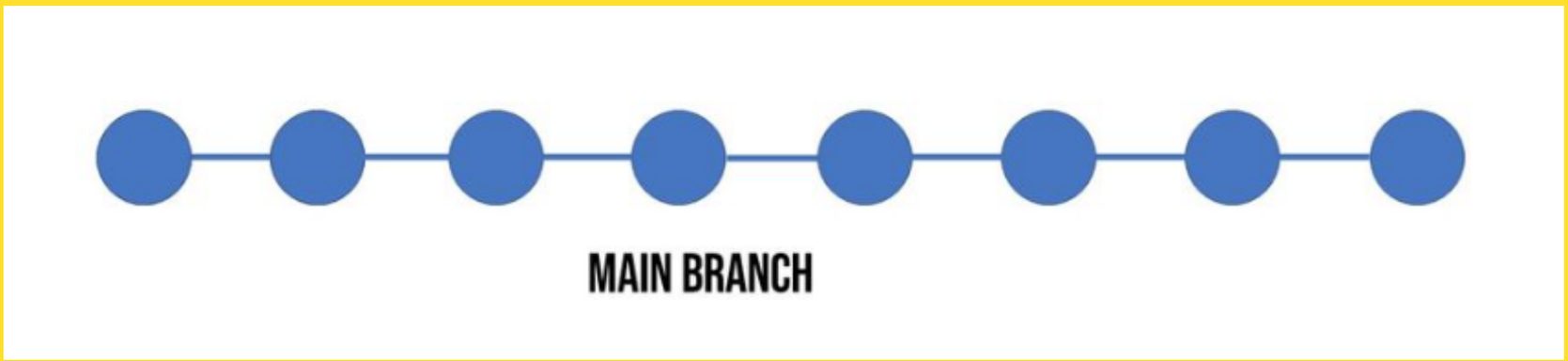
We have a repository with the main branch and then a feature branch. We've been working away on different things on this feature branch and have several commits. If we look at the graph of this, it will look something along the lines of this.



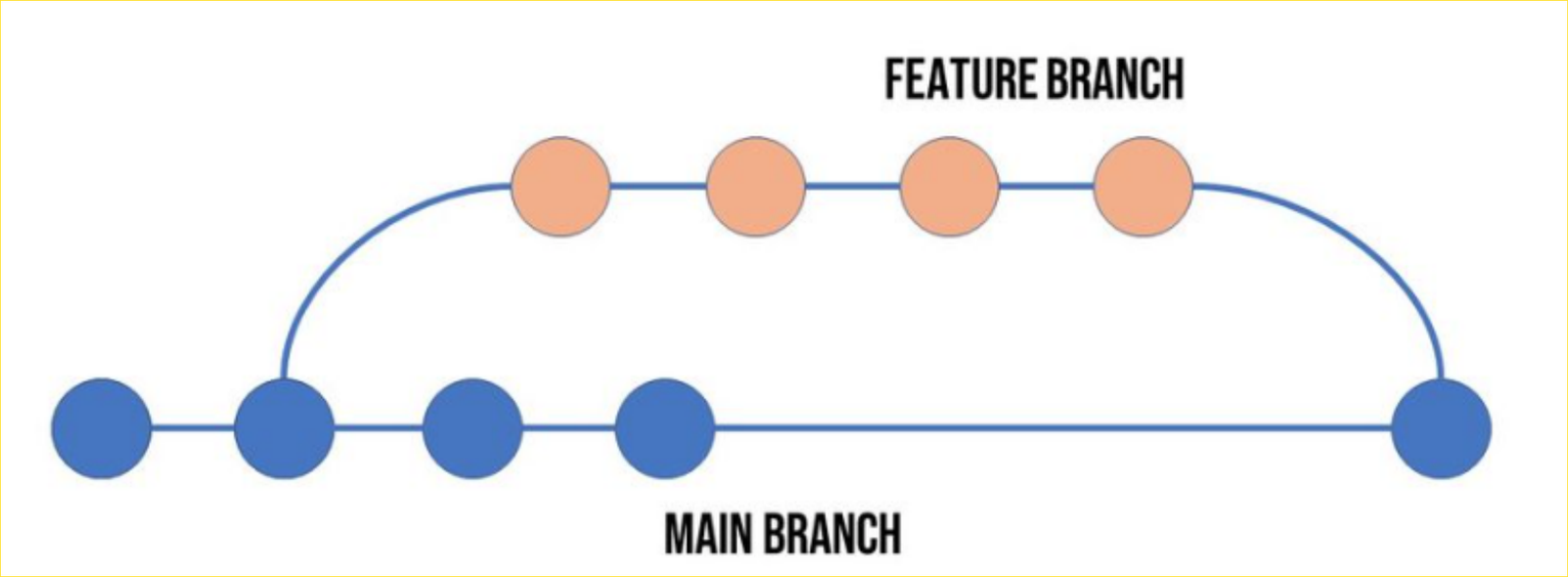
We want to bring all the commits and cool new features we've been creating from our feature branch into our main branch. But we want to make the history as linear as possible. So instead of doing a git merge we are going to use git rebase.

The commands we want to use are:
git checkout main
git rebase featurebranch

The first command makes sure we are in our main branch and the second command rebases that main branch to include the commits from feature.
Our git graph log would look like this:



You can see our history has been rebased into the main branch in a linear way.
If we'd used the git merge command instead this is what our repository history would look like:



Using the git rebase command gives you a much clearer history, especially if you have a large team of developers working on different features and bugs. It can become a real message if you are only using git merge. The history really becomes hard to read and understand.
Git rebase isn't always the right option though, as it does rewrite history. It's important as a team you understand the right use case for git rebase over git merge and use it appropriately.

For more visit:
https://www.techielass.com/git-rebasing/

12 Days of Git

DAY 10
GIT STASHING

@arjun-panwar

What is a git stash?

The git stash command temporarily stashes (shelves) any changes you've made to your working branch so you can work on something. When you are ready you can come back to the stashed changes and re-apply them later.
It's worth nothing that any stash is done only to your local Git repository copy, stashes are not transferred to the remote location.

Where is git stash saved?

When you issue the git stash command, they are stored inside a file called stash inside the .git/refs folder.
You can find all your stashes by using the command git stash list.
If you drop or clear any stashes it will remove it from the stash list, but you might still have some unpruned data within your local Git repository. So that is something to be mindful of.

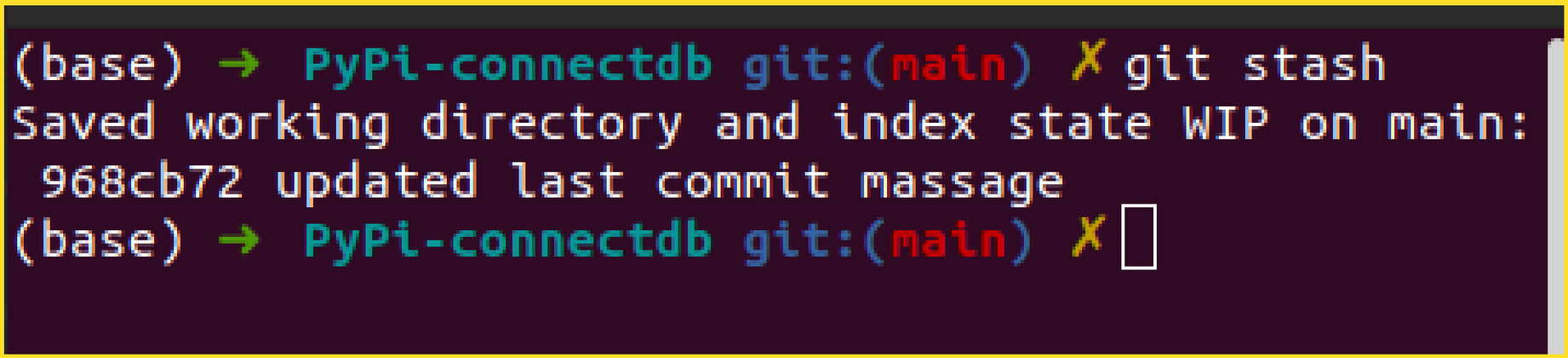
How do I use git stash?

The sequence of events for using git stash are as follows:

1. Save changes to branch 1
2. Run **git stash**
3. Check out branch 2
4. Work on the bug or feature in branch 2
5. Commit and push branch 2 changes to remote
6. Check out branch 1
7. Run **git stash pop** to get your stashed changes back

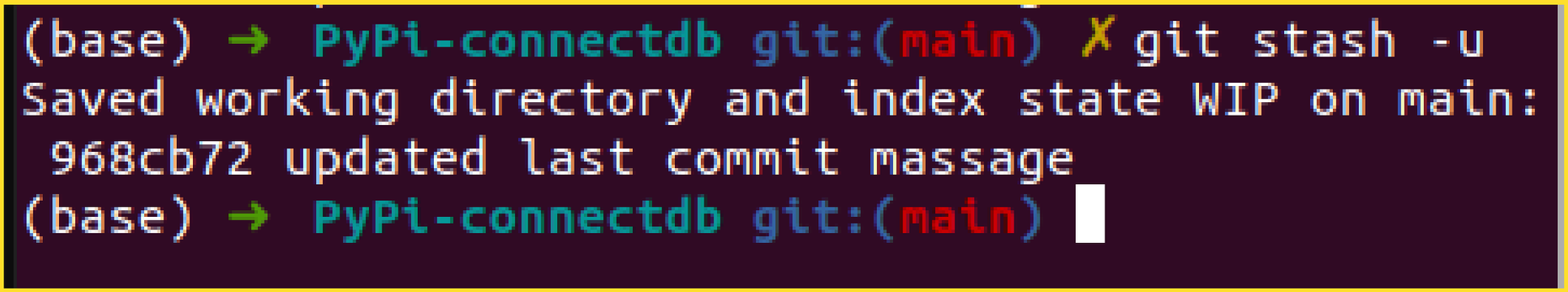
How to create a stash

Now we've looked at what stash does, where it stores information and the sequence of events when we are using. Let's take a look at how to use the commands in detail.
If we have unsaved, uncommitted changes within a branch we are working on we can just type **git stash**



This command will store or stash the uncommitted changes, either staged or unstaged files, and will overlook untracked or ignored files. If you wanted to stash untracked files, then you could use the command:

git stash -u or **git stash --include-untracked**



Alternatively, if you want to stash untracked files and ignored files you can use the command:

git stash -a or **git stash --all**

If there are specific files you want to stash then you can use the command:

git stash -p or git stash --patch

List git stashes

You can use the command **git stash list**

```
(base) ➜  PyPi-connectdb git:(main) git stash list

[2]  + 17759 suspended  git stash list
(base) ➜  PyPi-connectdb git:(main)
```

```
stash@{0}: WIP on main: 968cb72 updated last commit massage
stash@{1}: WIP on main: 968cb72 updated last commit massage
(END)
```

Managing multiple stashes

From the above list we can see we don't have much context as to what each of those stashes are. It's ***recommended best practice when using stash to write a save message.*** Your command would look like this:

```
git:(main) git stash save "test stash msg"
```

When you list out your stashes you'll now have more information refer to.

Retrieve git stash changes

Now you have something stashed away, how do you retrieve it? There are two commands that you could use, **git stash apply** or **git stash pop**.

Both of the commands reapply the changes that were stashed. There is a slight difference between what they do.

- Git stash apply reapplies the changes
- Git stash pop removes the changes from stash and reapplies them to the working copy

Git stash apply allows you to reapply the stashed changes more than once. You can only use the Git stash pop command once.

Apply or Pop a specific stash

To apply a stash and remove it from the stash list, run:

git stash pop stash@{n}

To apply a stash and keep it in the stash cache, run:

git stash apply stash@{n}

(note that in some shells you need to quote "stash@{0}", like zsh, fish and powershell).

git stash apply n

git stash pop n

Clear stash

You can clear out a specific stash by using the command:

git stash drop stash@{1}

{1} is stash index number

```
stash@{0}: WIP on main: 968cb72 updated last commit massage
stash@{1}: WIP on main: 968cb72 updated last commit massage
(END)
```

Or if you want to get rid of all stashes then you can use the command:

git stash clear

For more visit:
https://www.techielass.com/git-stashing/

12 Days of Git

DAY 11
GIT CONFLICTS

@arjun-panwar

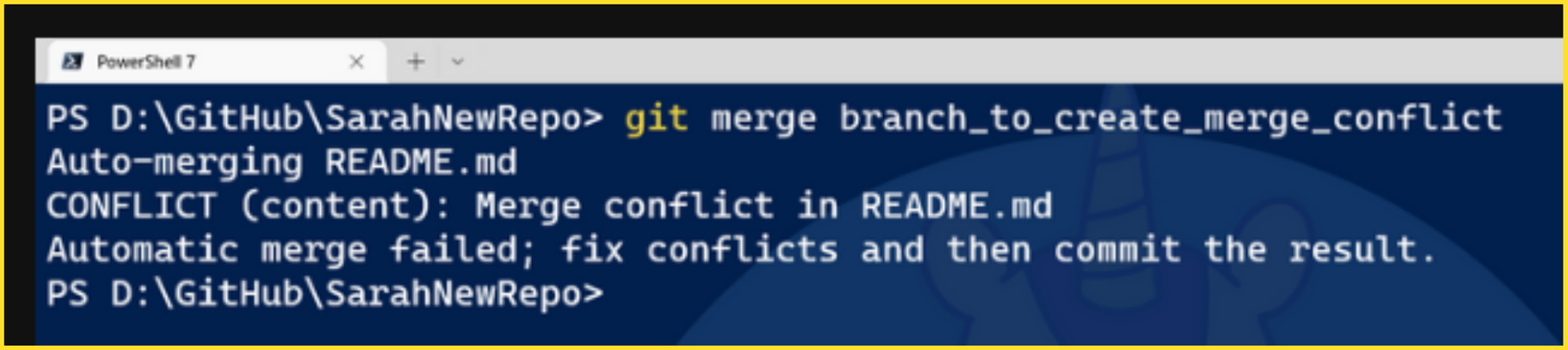
What are Git conflicts?

As we've seen Git can handle merges automatically most of the time. Git conflicts arises when two separate branches have made edits to the same line within a file or even when a file has been deleted in one branch and changed in another.

When teams of people are working on different branches, features, bugs this is when conflicts are likely to happen. They are almost part of the collaborating nature of working with Git and source control.

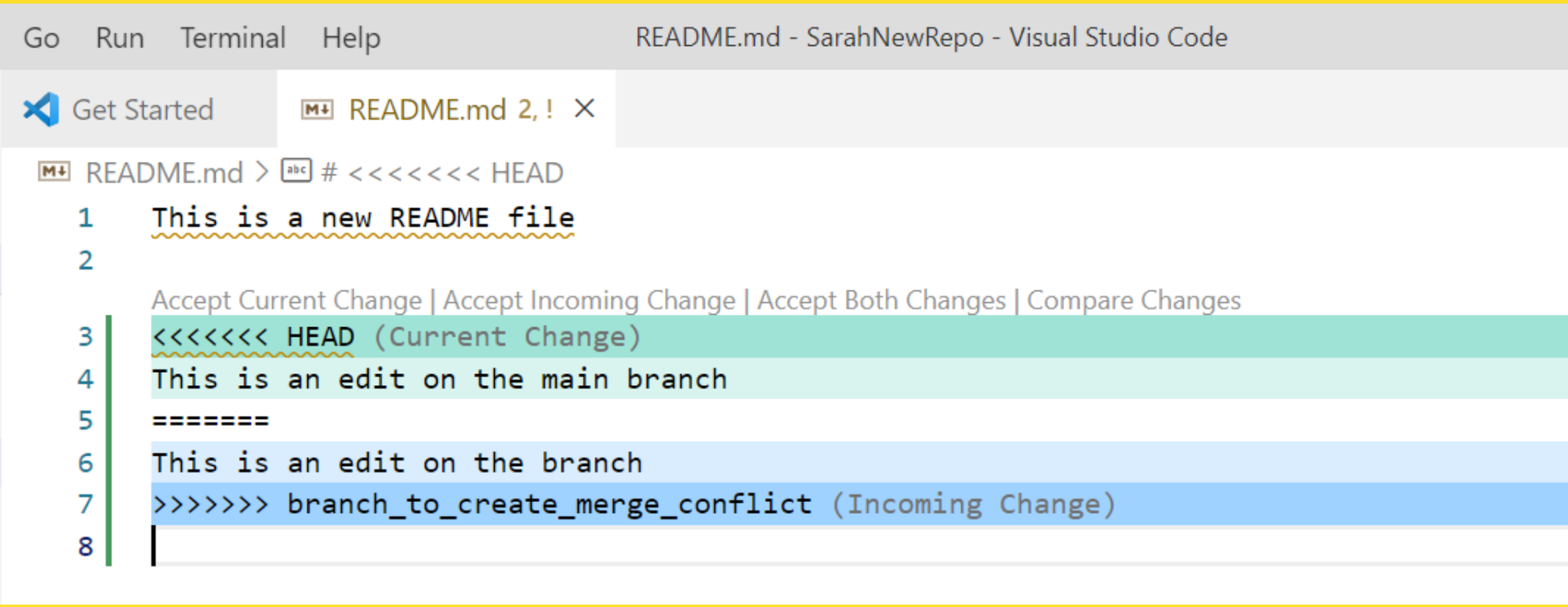
How do you resolve a git merge conflict?

Generally, you'll get information back from Git when there is a merge conflict on what you should do to resolve it.
Below is an error message when I tried to merge a branch into my main branch:



I deliberately made changes to the README on both branches to trigger this error, but you can see the error message has told me there is an issue with the README file.

If we switch over to Visual Studio Code and open that README file we can see more information as to what is causing the error:



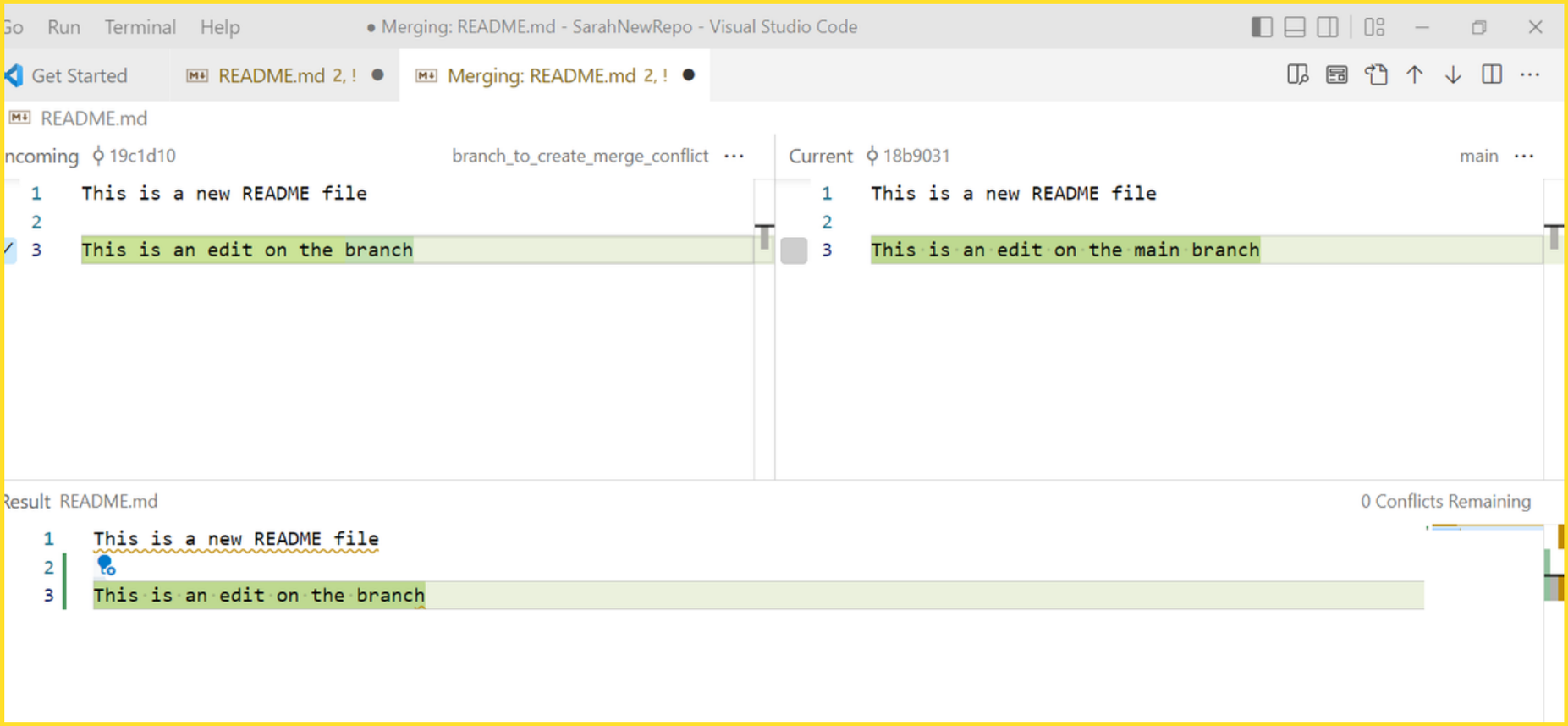
As we can see in here Git has added some syntax for us, we have "less than" characters and "greater than" characters. There are 7 of each, and you can use these to search through your editor quickly if you have a lot of edits to be made.

Within this example we have two sections:

- The "less than" characters denote the current branch's edits, and the equal signs denote the end of the first section.
- The second section is where the edits are from where we attempted to merge. This time it starts with the equal signs and ends with the "greater than" signs.

It's up to you as the engineer, or merger to decide what stays and what goes. Within Visual Studio Code you also have the option to use the "Merge Editor", which gives you an even greater look at what is conflicting, and helps you correct it.

If I open the Merge Editor it can help me either accept the incoming merge, the current one or both merges. I want to accept the change from the branch, so I put a tick next to that and accept the merge.



There are other ways you might encounter a merge conflict, within GitHub for example. And there is great documentation by GitHub on how to deal with that.

Dealing with conflicts will always be tricky, deciding what to keep, what to delete, what to merge together. You as the merger or owner or even as a team need to decide what is best.

Good clear pull requests into repositories is needed, keep your pull requests simple, fix one bug or add one feature at a time. Give good explanations of what you are doing, etc That way if there is a merge conflict whoever is trying to decide what to do will have the right context and make their job easier.

For more visit:
https://www.techielass.com/git-conflicts/

12 Days of Git

DAY 12
SUBTREES AND SUBMODULES

@arjun-panwar

What is a git submodule?

Git submodules allow you incorporate and track version history of external code. Sometimes what you are working on, will depend upon external code. You could just copy and paste that external code into your main repository and use it. But there is the obvious disadvantage here of having too manually maintain that code and grab updates when they arise.

You could use something like a package management system to help maintain the code, Ruby Gems or NPM. But again, there are downsides to this approach as well.

Neither of these methods incorporate methods to track edits and changes of the external repository.
A Git submodule is a recording within a Git repository that points to a specific commit in another external repository.
Submodules are static and only track specific commits, they don't track git refs or branches. And don't automatically update when the external repository is updated.

When should I use a git submodule?

The use cases for Git submodule are:

- When you have a component, you rely on that isn't updates very often and you want to track it as a dependency.
- When you are delegating a piece to a third party, and you want to integrate their work at a specific release point.
- When you rely on an external component that changes to fast, and you want to lock the code to a specific commit or point in time.

What is git subtree?

Git subtree lets you nest a repository inside another as a sub directory.

It's worth mentioning that there is also a merging strategy called subtree. So be careful with this term and ensure when talking or referencing subtree.

Git submodule vs git subtree

There are a lot of articles that say submodules aren't something you should use, and subtrees are the way forward. There are some benefits of subtree over submodule. But subtree has its own disadvantages. If we look at why people prefer subtree over submodule:

- It's an easier workflow.
- There is nothing to learn when using subtree, they can ignore the fact they are using subtree to manage dependencies.
- Subtree doesn't add metadata files like submodule does.

The drawbacks of subtree though are:

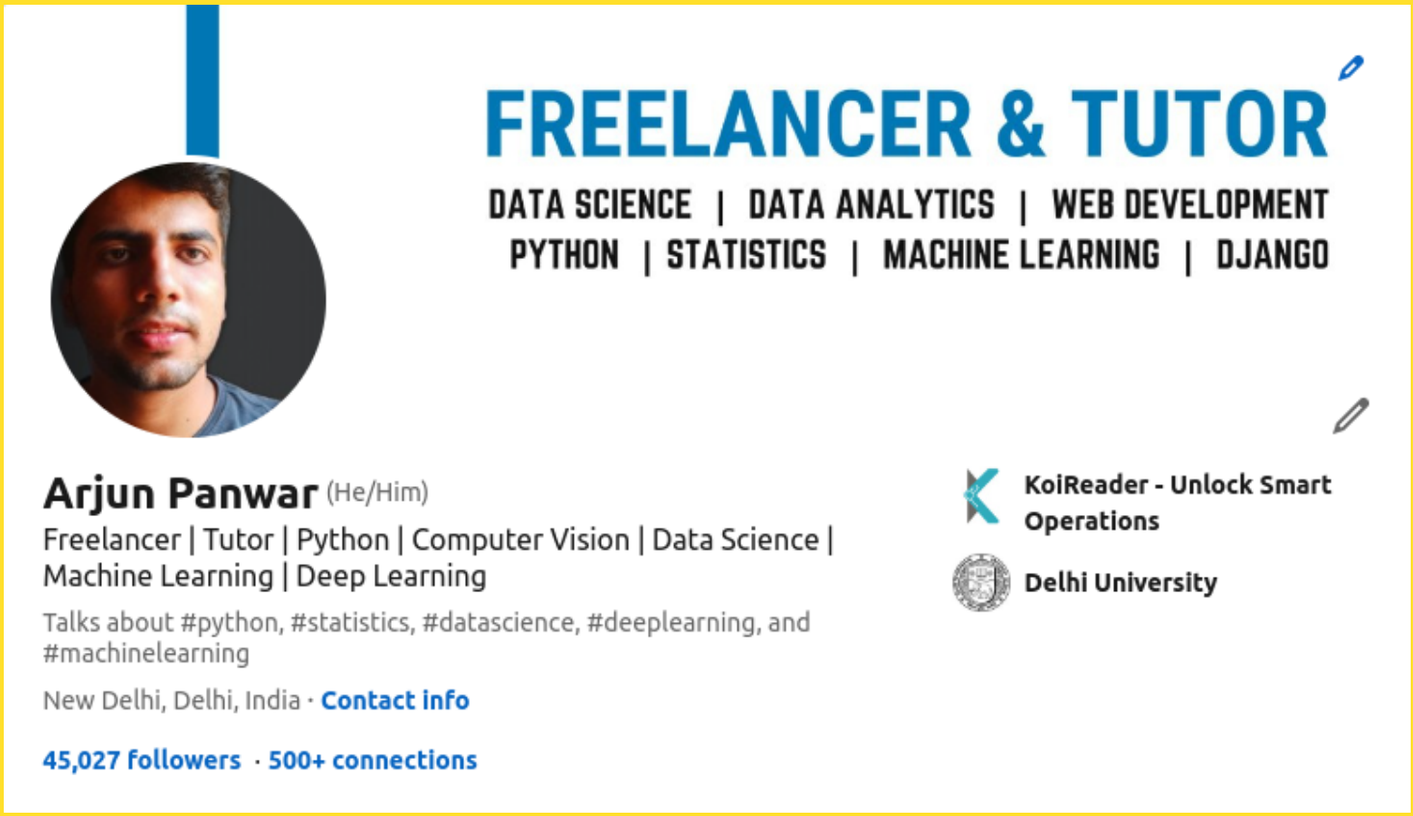
- Contributing back to the original code is more complicated.
- You need to be careful not to mix commits with your project and the third-party code.

Both have their use cases but from the research I have done the preference seems to be git subtree, as it overcomes the drawbacks of git submodule.

For more visit:
https://www.techielass.com/subtrees-and-submodules/

Do you want to learn Python, SQL, Django, Machine Learning, Deep Learning, and Statistics in one-on-one classes

Drop me a message on LinkedIn to discuss your requirements



@arjun-panwar