Twitter Thread by Sunrit Jana ■

Sunrit Jana ■
@JanaSunrise

Git is used in a ton of places. Companies, SaaS, open-source projects use it to work, collaborate and maintain their projects.

All the git commands you would ever need, and also to get you started. Here's a thread for you.

Let's explore all the commands.

↓

This thread is continuation for my thread on Git for everyone, where I explained all about git.

If you haven't checked that out, and if you're willing to, Please do. Appreciate it, Here's the link. ↓

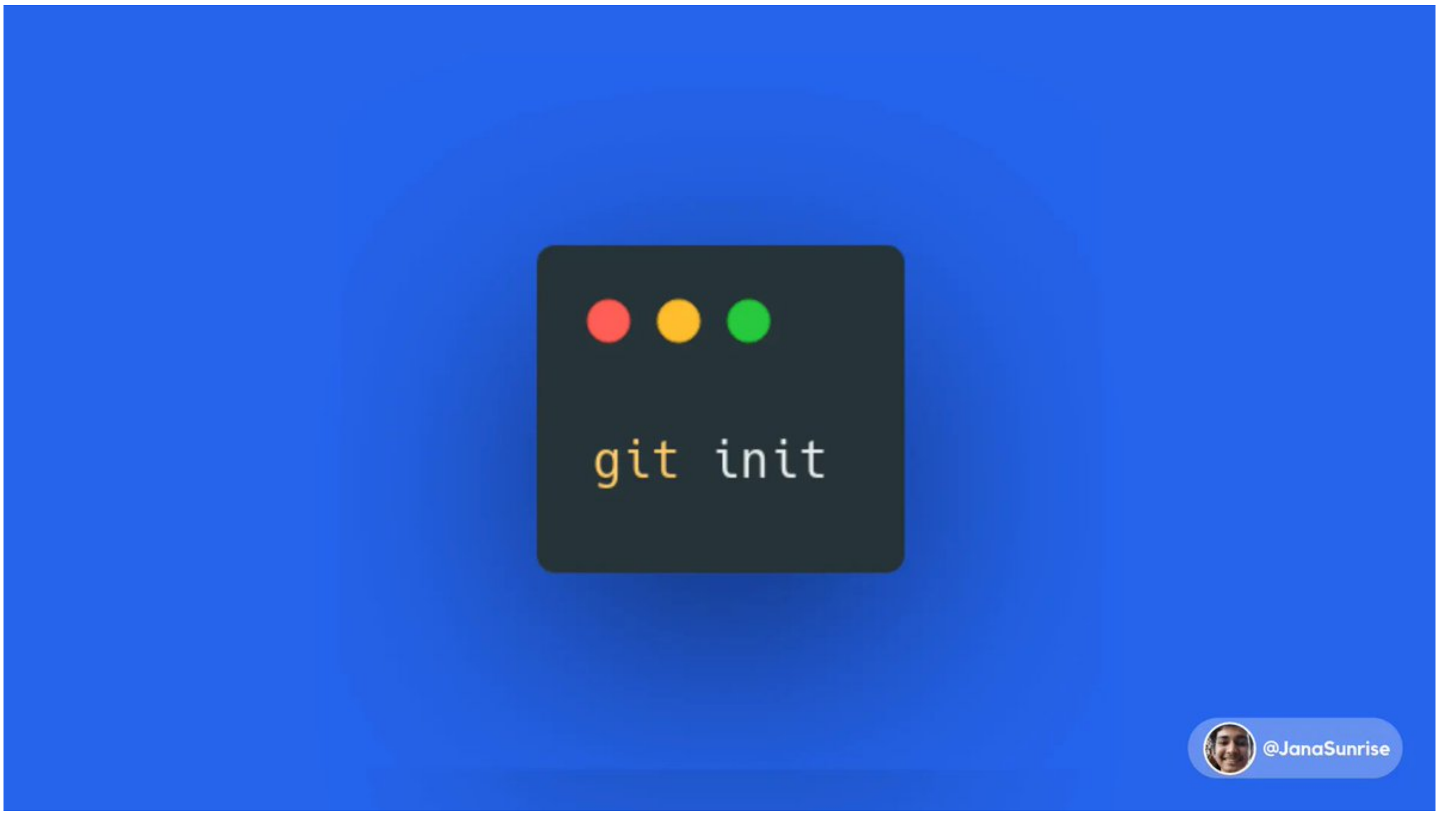
https://t.co/YBQDXSF9ga

> Git is an incredible tool made for easily managing your projects, it's history and version control \U0001f525
>
> It is really easy to learn, yet can save you a lot of hassle and time managing projects.
>
> Here's a mega thread to walk you through all about git. Let's go! \u2193
>
> — Sunrit Jana \U0001f680 (@JanaSunrise) November 5, 2021

1/ Initializing git in project

This is the command, you would always need. This initializes git, along with it's structure for tracking and managing files, folders and code.

You cannot use any other command, if not initialized.

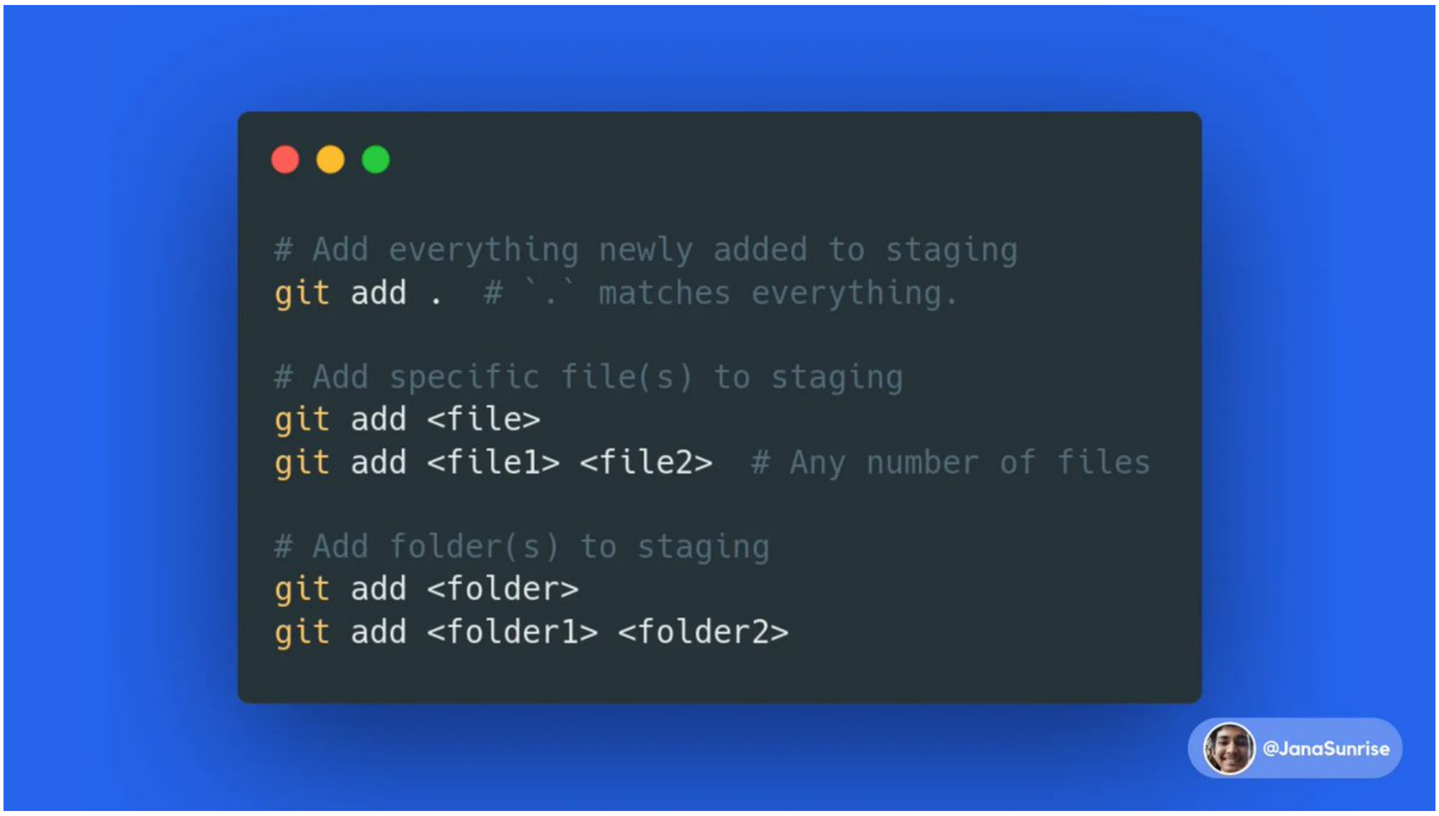


```
git init
```

2/ Adding files to staging

Now, we've initialized repo, and Created files. We have to move the files to staging, before commiting them.

Turns out, there are a lot of ways to do it.

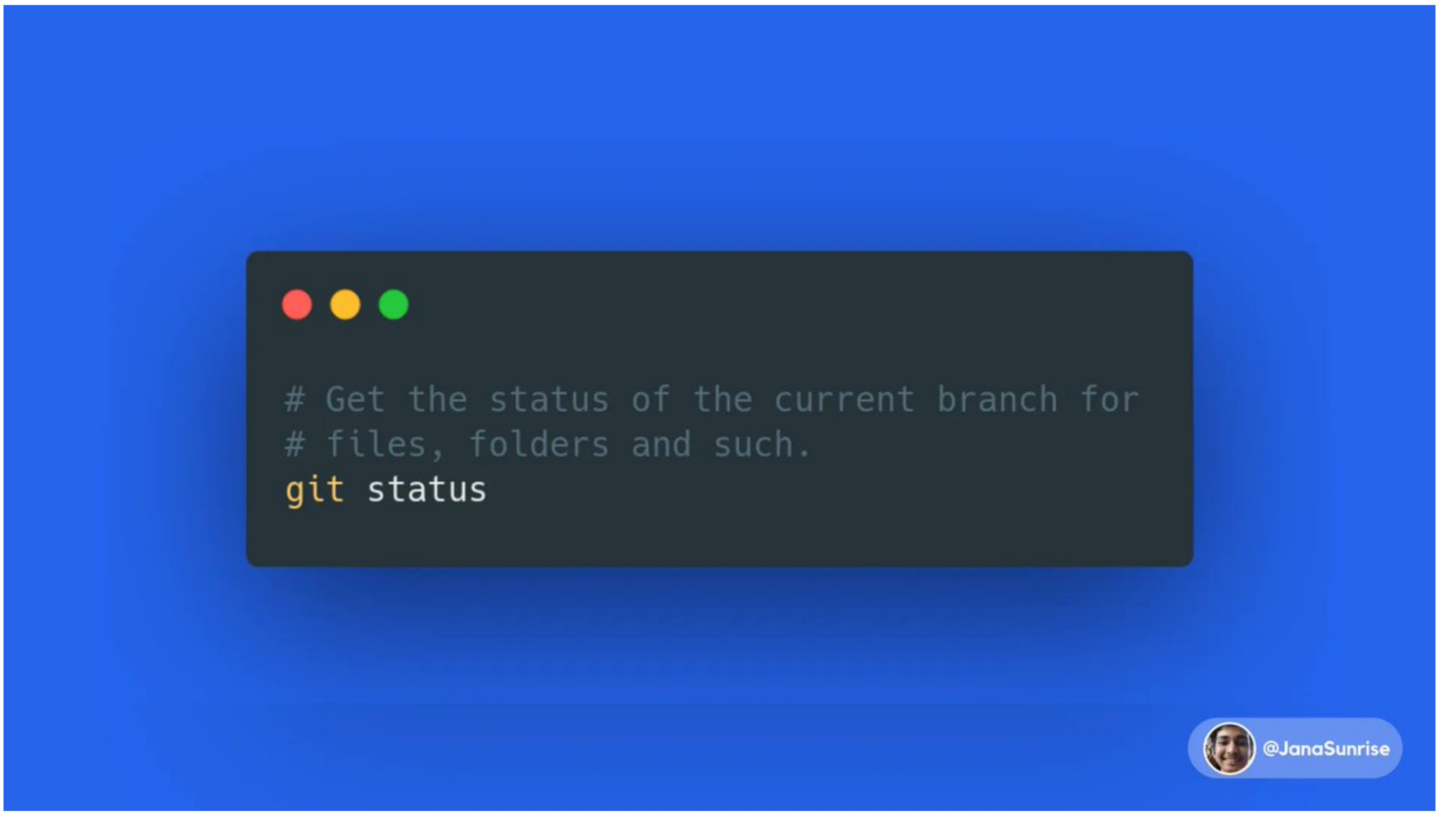


```
# Add everything newly added to staging
git add .  # `.` matches everything.

# Add specific file(s) to staging
git add <file>
git add <file1> <file2>  # Any number of files

# Add folder(s) to staging
git add <folder>
git add <folder1> <folder2>
```

3/ Track changes, and Get info about staging stage

Once we've added files, folders and other things to staging, We have the need to ensure everything's correct, or check if there's something wrong, or keep track.

Git allows to display all the changes, additions, and deletions.

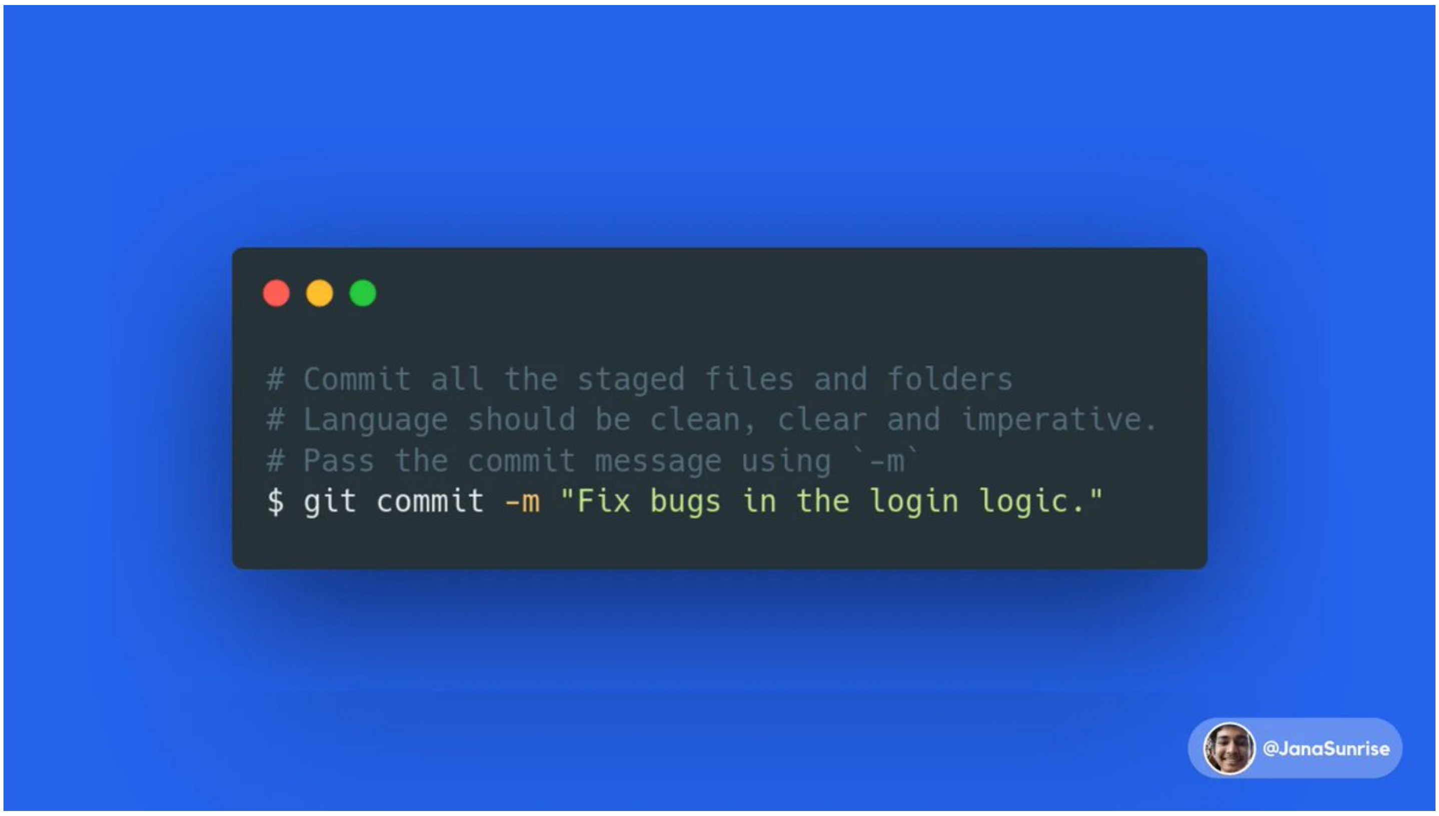


```
# Get the status of the current branch for
# files, folders and such.
git status
```

4/ Commit files, and create a log

We're good to commit now! When we commit, It creates a commit object with all info mentioned in previous thread.

A commit message is necessary to properly describe it. The language should be imperative, Not more than 50 characters.

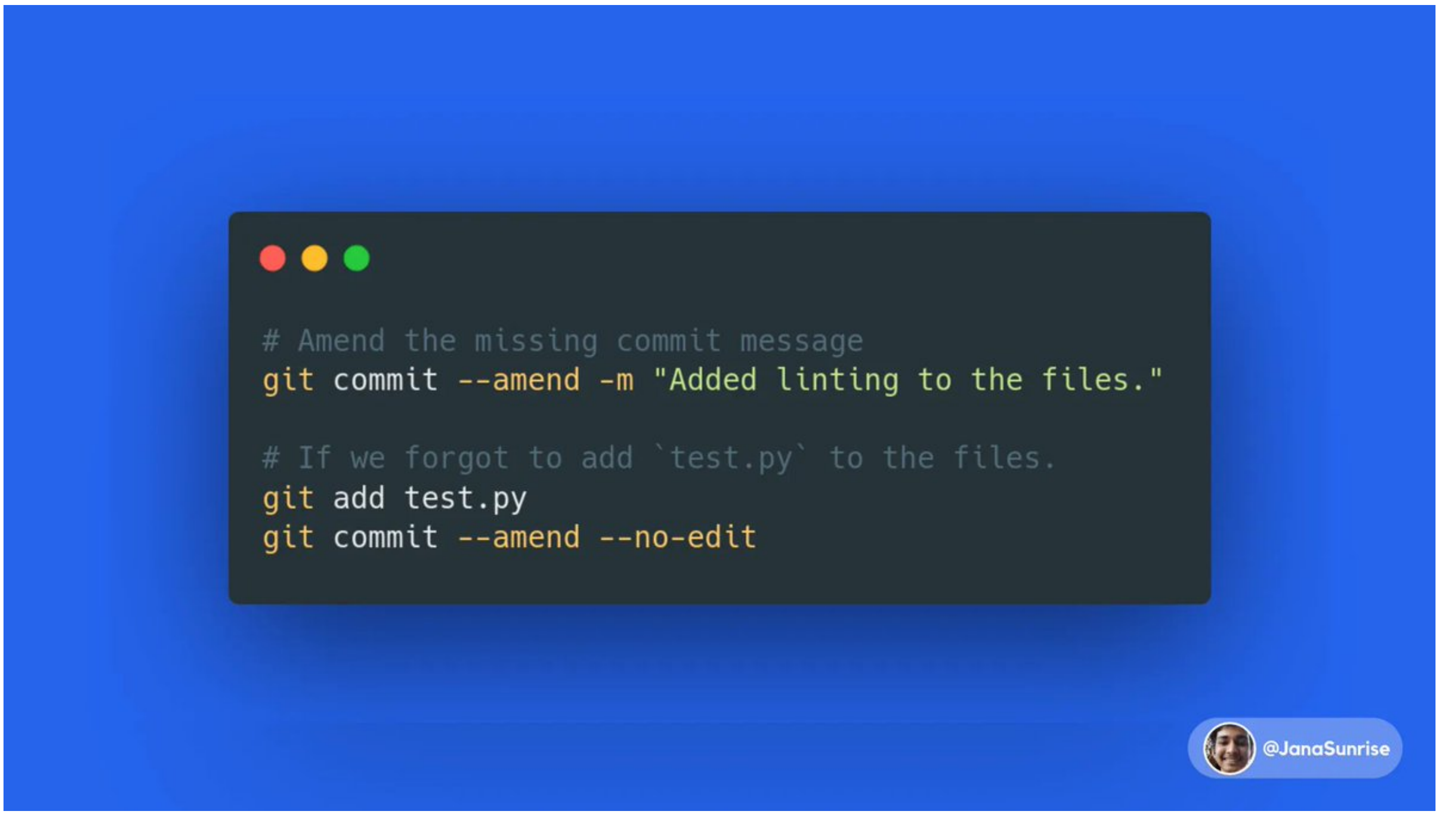


```
# Commit all the staged files and folders
# Language should be clean, clear and imperative.
# Pass the commit message using `-m`
$ git commit -m "Fix bugs in the login logic."
```

5/ git commit --amend

Oops! What if we missed out something in the commit message, or forgot to stage a file?

Git commit with amend flag comes to the rescue. Here is how you can use it!

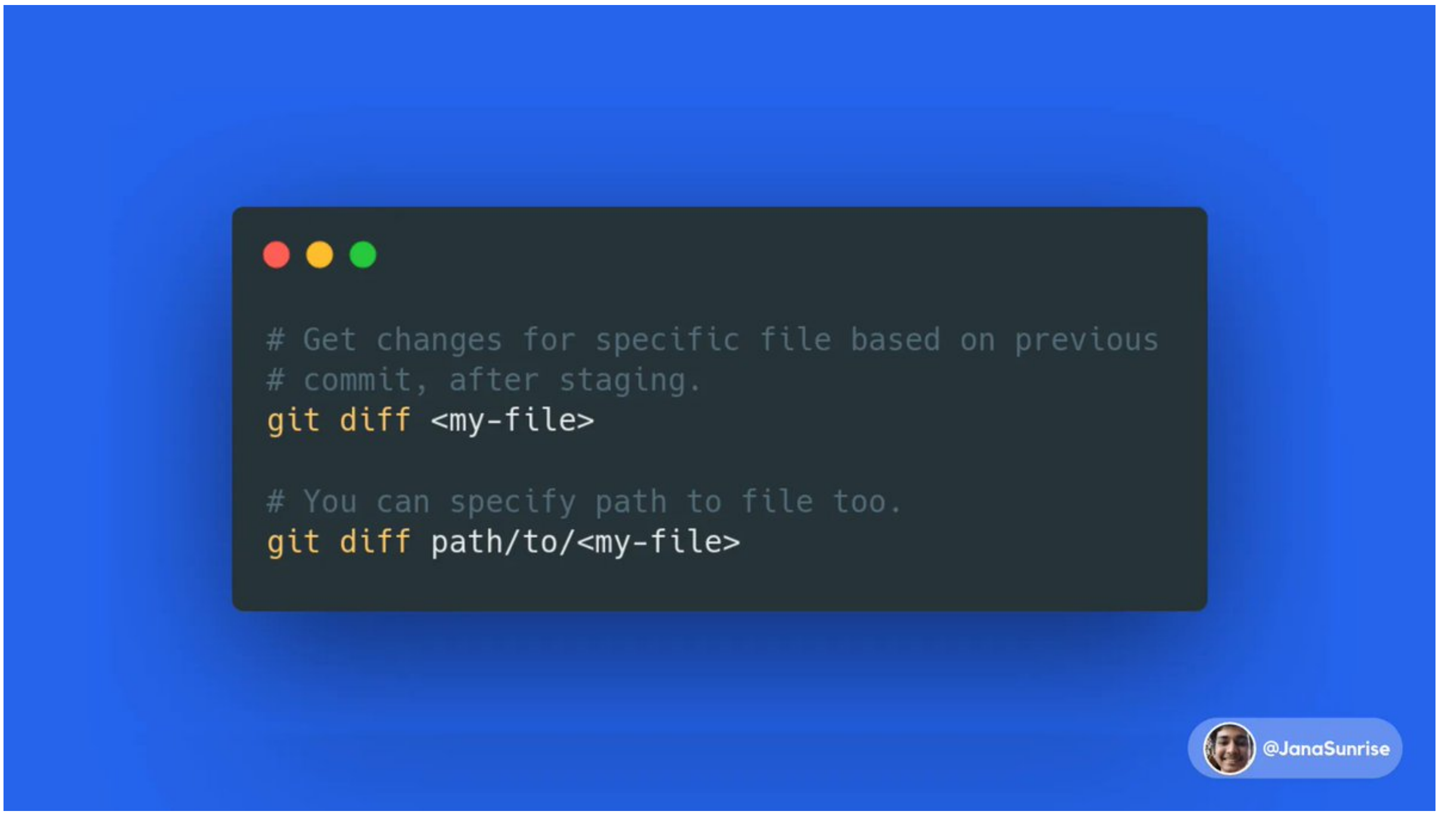


```
# Amend the missing commit message
git commit --amend -m "Added linting to the files."

# If we forgot to add `test.py` to the files.
git add test.py
git commit --amend --no-edit
```

6/ Getting changes for specific files after staging

This is what git is all about. Maintaining history, and list of changes. Hence, Git allows us to preview the changes in file specified based on what's changed since previous commit.

Here is how!

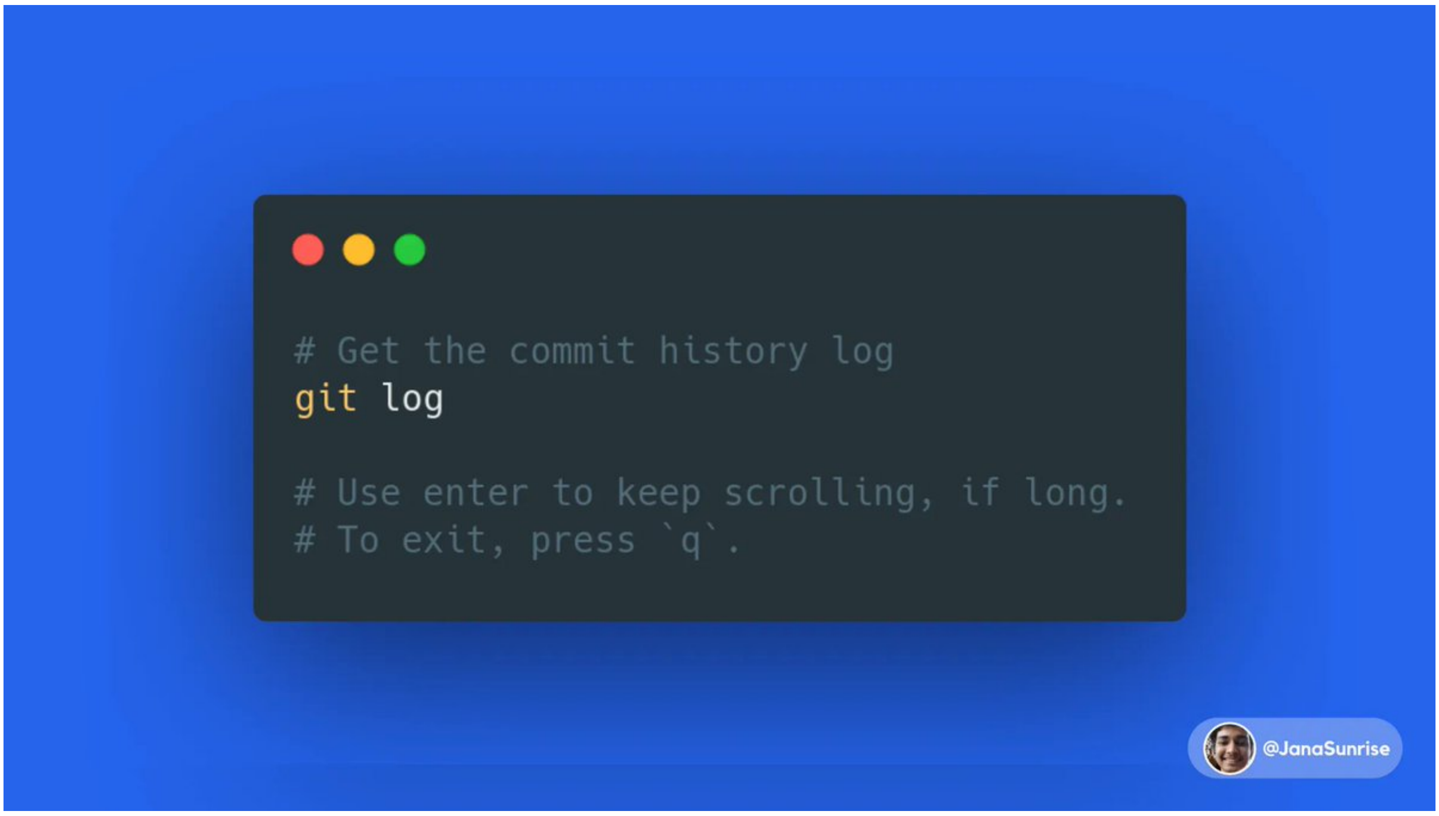


```
# Get changes for specific file based on previous
# commit, after staging.
git diff <my-file>

# You can specify path to file too.
git diff path/to/<my-file>
```

7/ Getting commit history!

As git makes commit objects with all metadata, We should be able to access it! And, we can too. Just wonderful. It gives all metadata, we need, along with the Hash to revert/reset and whatever operation we have to perform.

Here is the command!



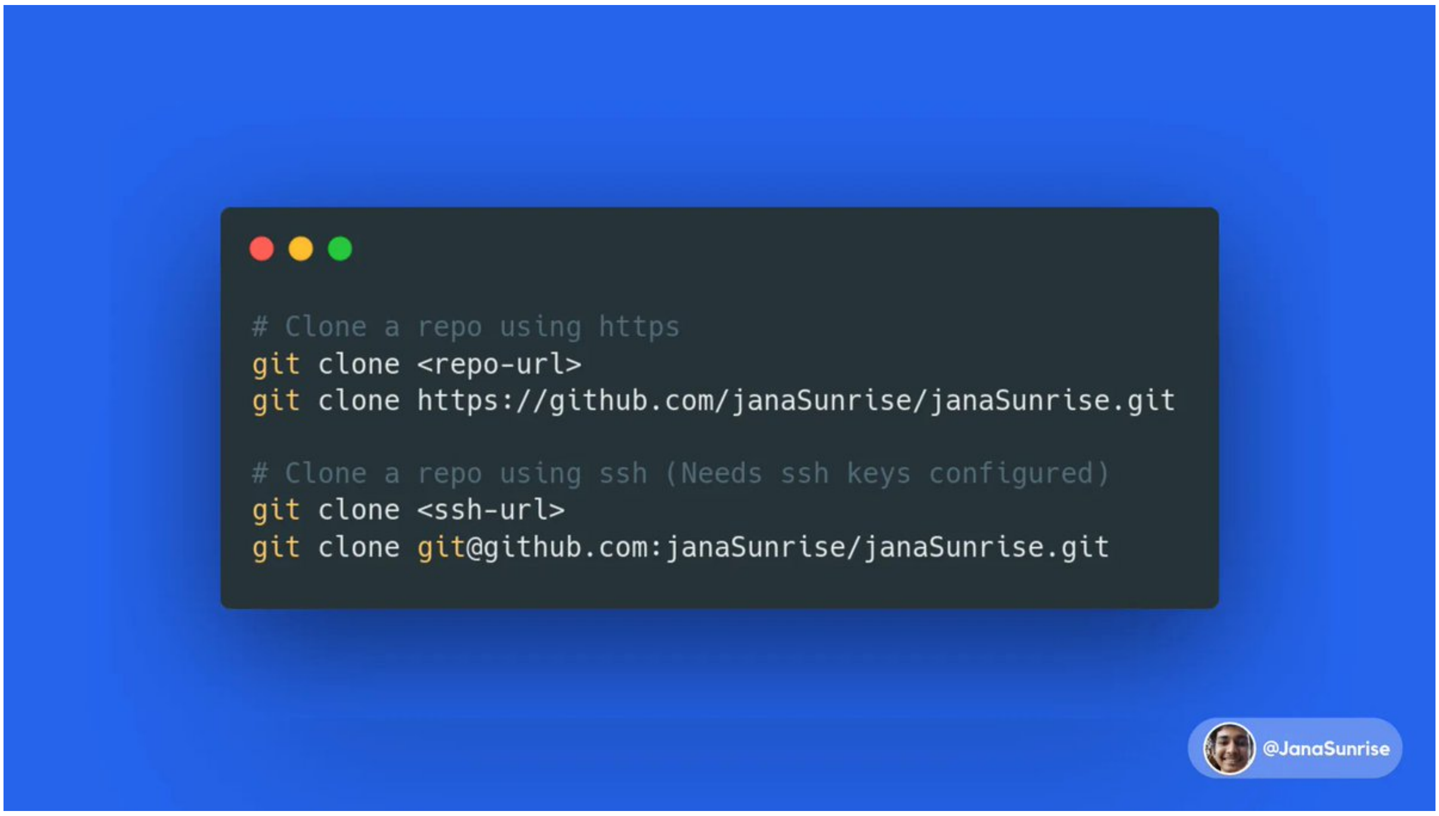
```
# Get the commit history log
git log

# Use enter to keep scrolling, if long.
# To exit, press `q`.
```

8/ Cloning a repository

We always need the project to be present locally when we want to work. Git provides an extremely *easy* way to do that.

You need to have the URL to the remote repository along with permissions to clone.

Here's how,



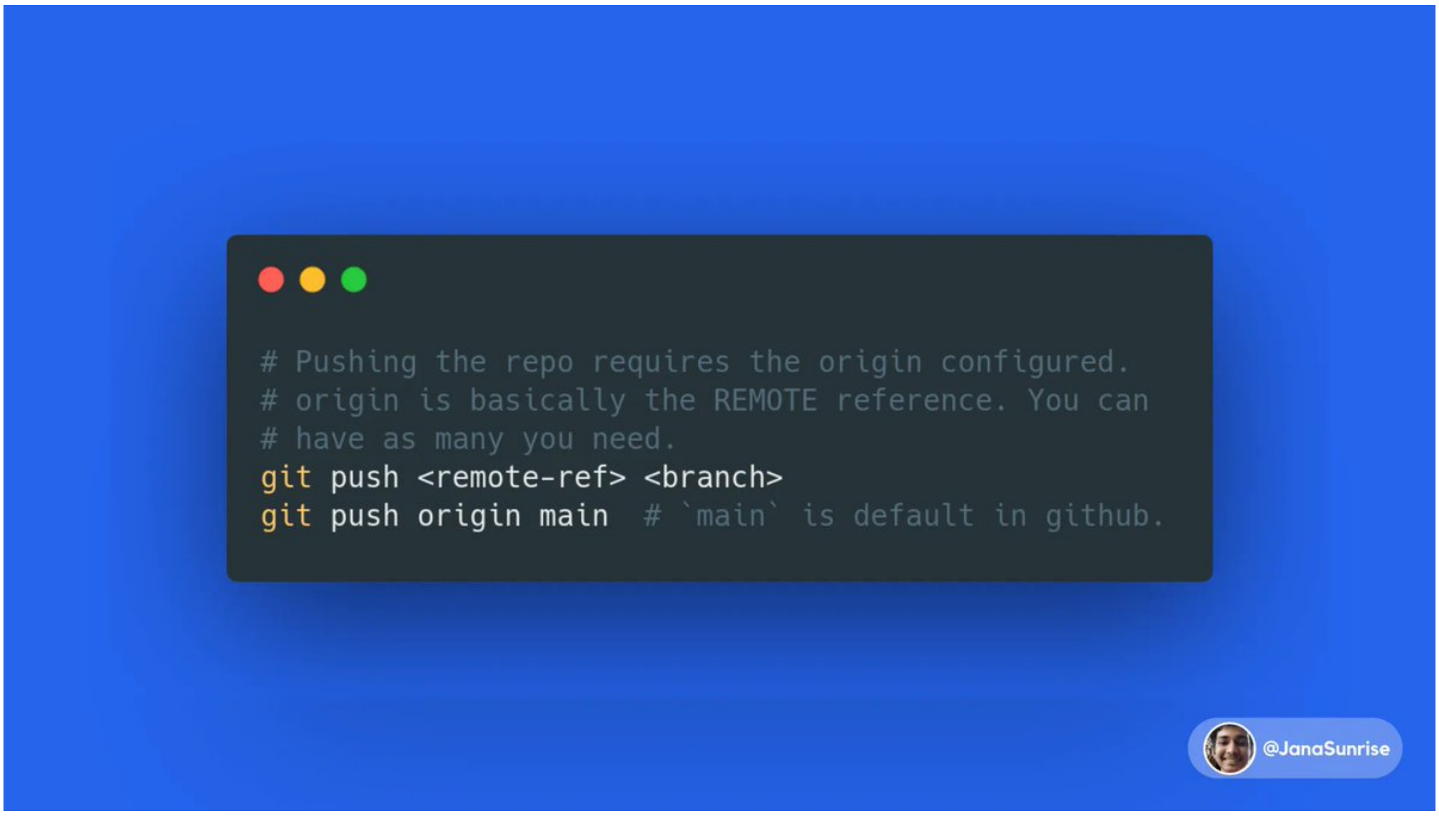
```
# Clone a repo using https
git clone <repo-url>
git clone https://github.com/janaSunrise/janaSunrise.git

# Clone a repo using ssh (Needs ssh keys configured)
git clone <ssh-url>
git clone git@github.com:janaSunrise/janaSunrise.git
```

9/ Pushing to a repository

Okay! We have our brand new repository. And we have created the basic project to get us started.

Now, we need to update the changes in remote, by pushing, and also enabling others to get your changes.

Here is how you can push,

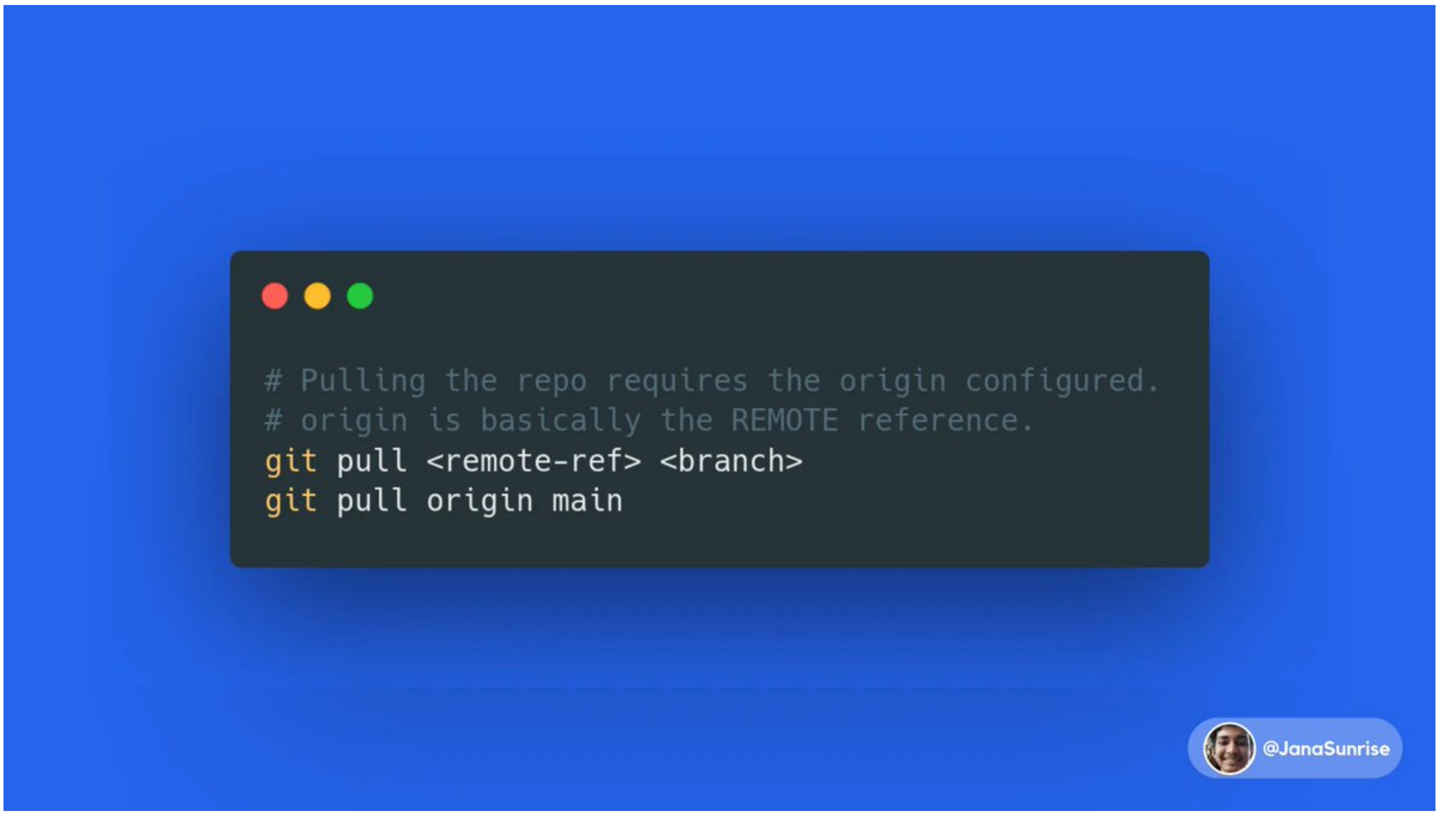


```
# Pushing the repo requires the origin configured.
# origin is basically the REMOTE reference. You can
# have as many you need.
git push <remote-ref> <branch>
git push origin main  # `main` is default in github.
```

10/ Pull changes from remote repo to local

Oh, so we already have the repo locally, but the changes are outdated. How do I pull the new changes and update my local branch to work further?

It's easy.

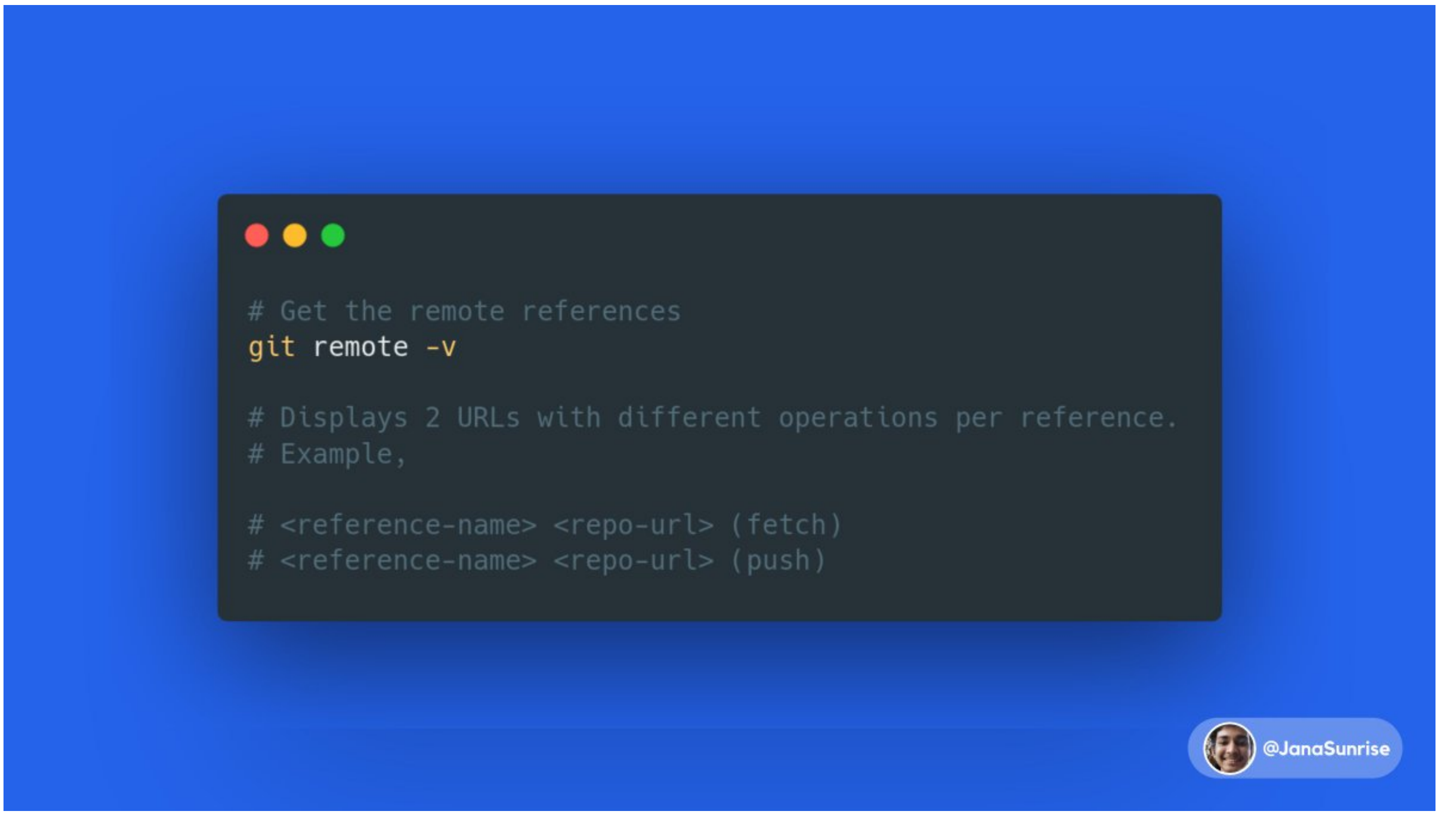


```
# Pulling the repo requires the origin configured.
# origin is basically the REMOTE reference.
git pull <remote-ref> <branch>
git pull origin main
```

11/ Getting configured remote repository paths

Working with remote repositories needs us Remote repositories configured. Git allows us to see the configured ones, and take actions based on it.

Here's how.

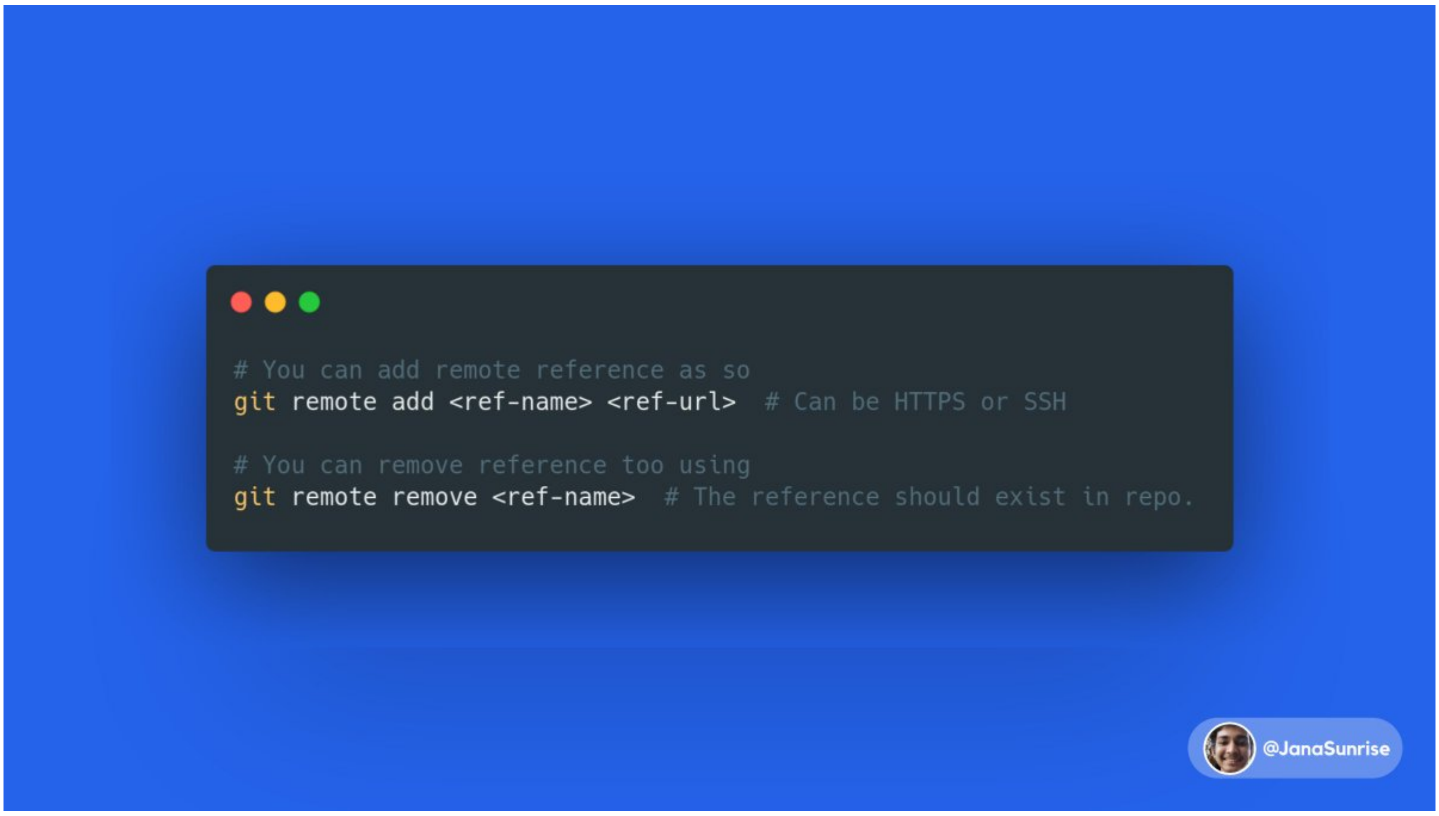


```
# Get the remote references
git remote -v

# Displays 2 URLs with different operations per reference.
# Example,

# <reference-name> <repo-url> (fetch)
# <reference-name> <repo-url> (push)
```

12/ Configuring and modifying remote repos

We also have the need to add remote repositories, to push to them, and have a reference, or delete old reference to repositories.

These are not that frequently used, but can be a lot of use when needed.

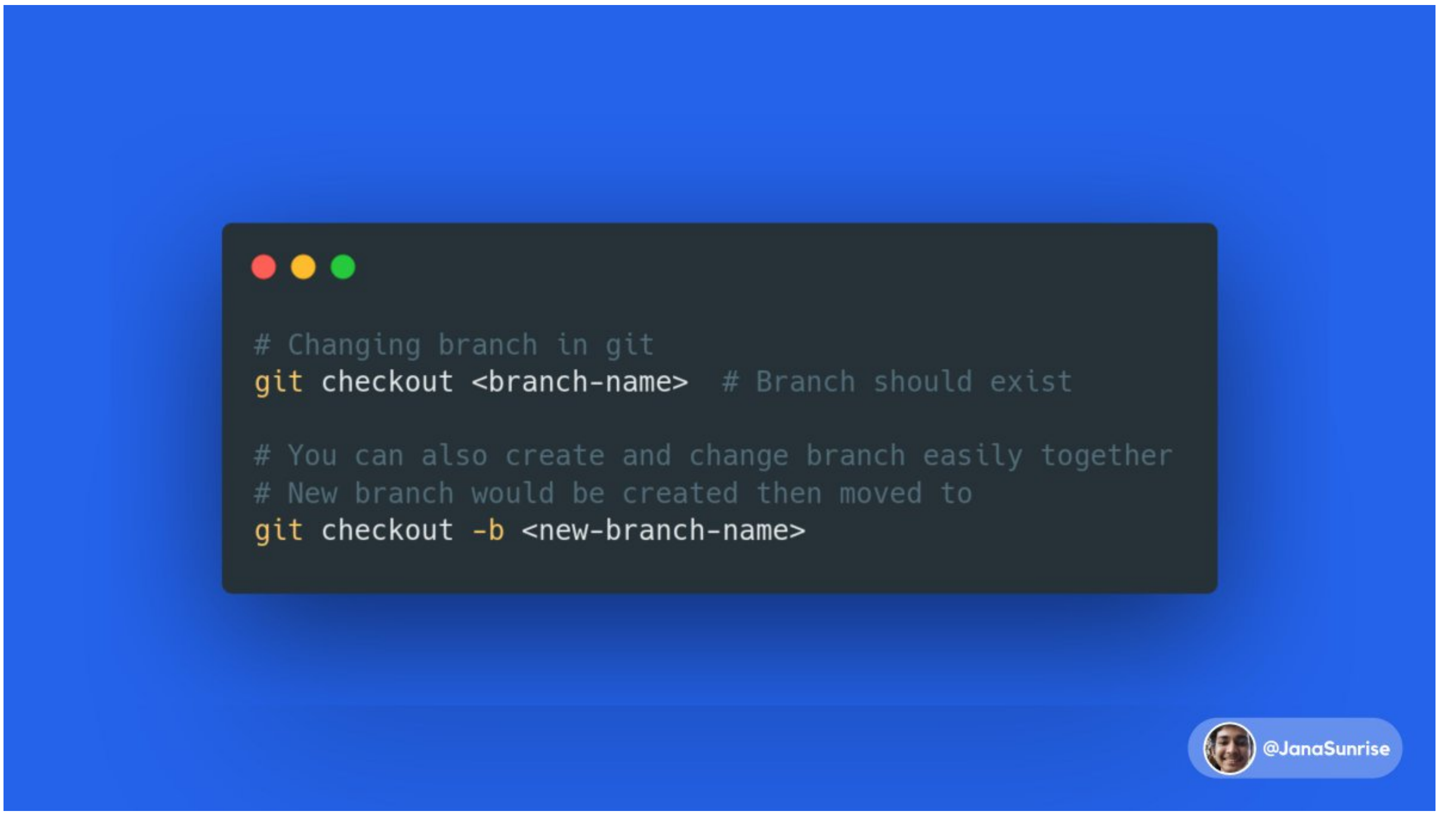


```
# You can add remote reference as so
git remote add <ref-name> <ref-url>  # Can be HTTPS or SSH

# You can remove reference too using
git remote remove <ref-name>  # The reference should exist in repo.
```

13/ Changing branch

Working in branches on specific features, fixes or any kind of changes is really frequent, and making PRs to review and merge them.

Changing branch is highly used. Here is how you can do that!



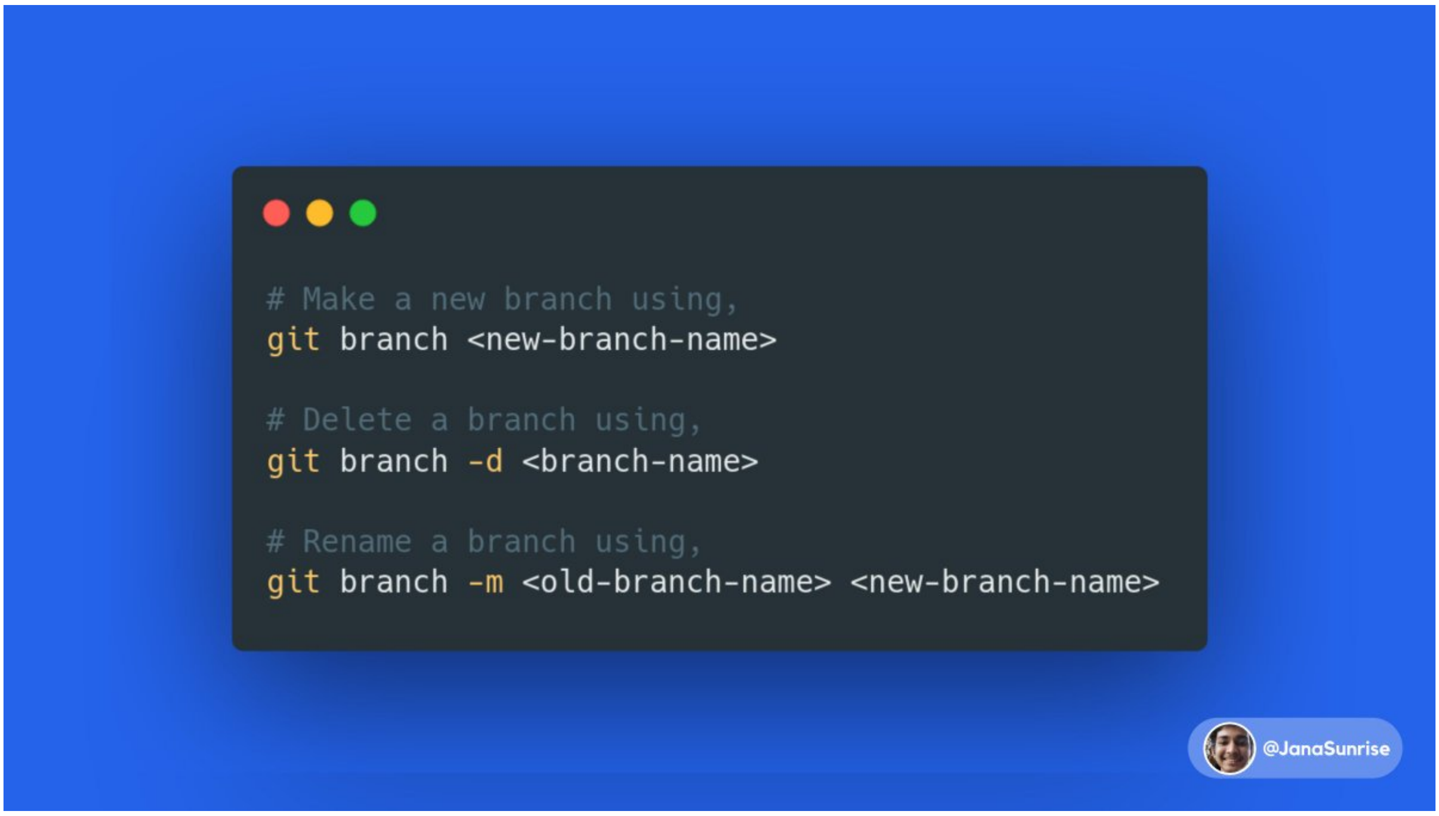
```
# Changing branch in git
git checkout <branch-name>  # Branch should exist

# You can also create and change branch easily together
# New branch would be created then moved to
git checkout -b <new-branch-name>
```

14/ Working with branches

Well, We can change branches. But what about operations?

Such as, creating branch to work in it, deleting a branch after work's done and merged, or renaming a branch to describe it better?

All these are really simple.



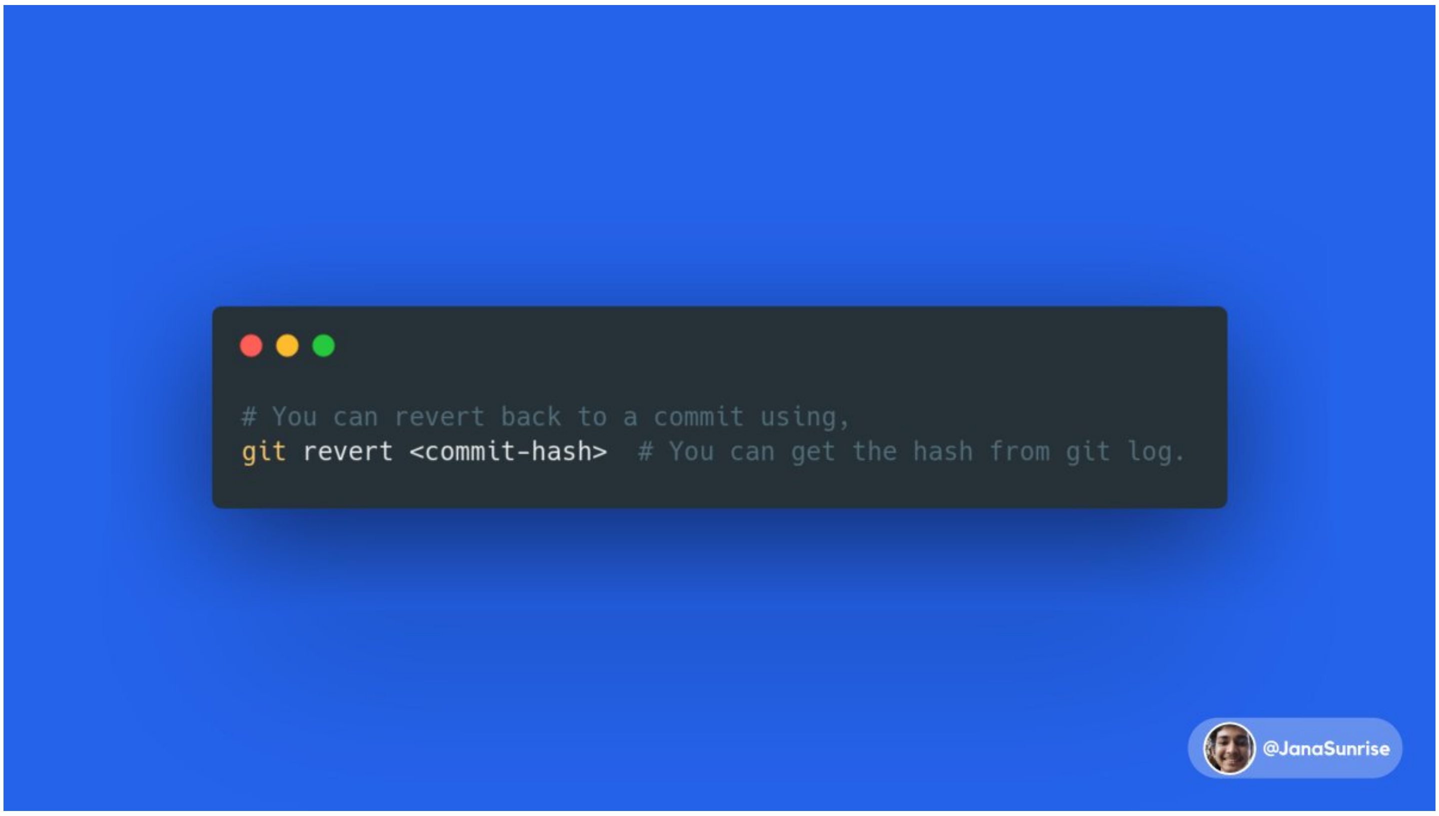
```
# Make a new branch using,
git branch <new-branch-name>

# Delete a branch using,
git branch -d <branch-name>

# Rename a branch using,
git branch -m <old-branch-name> <new-branch-name>
```

15/ Revert useless commits

A lot of times, we might have commited somethings, We didn't intend, and We wish to go back to previous state (or commit).

Git allows to do that, using the `revert` subcommand.

Here's how,

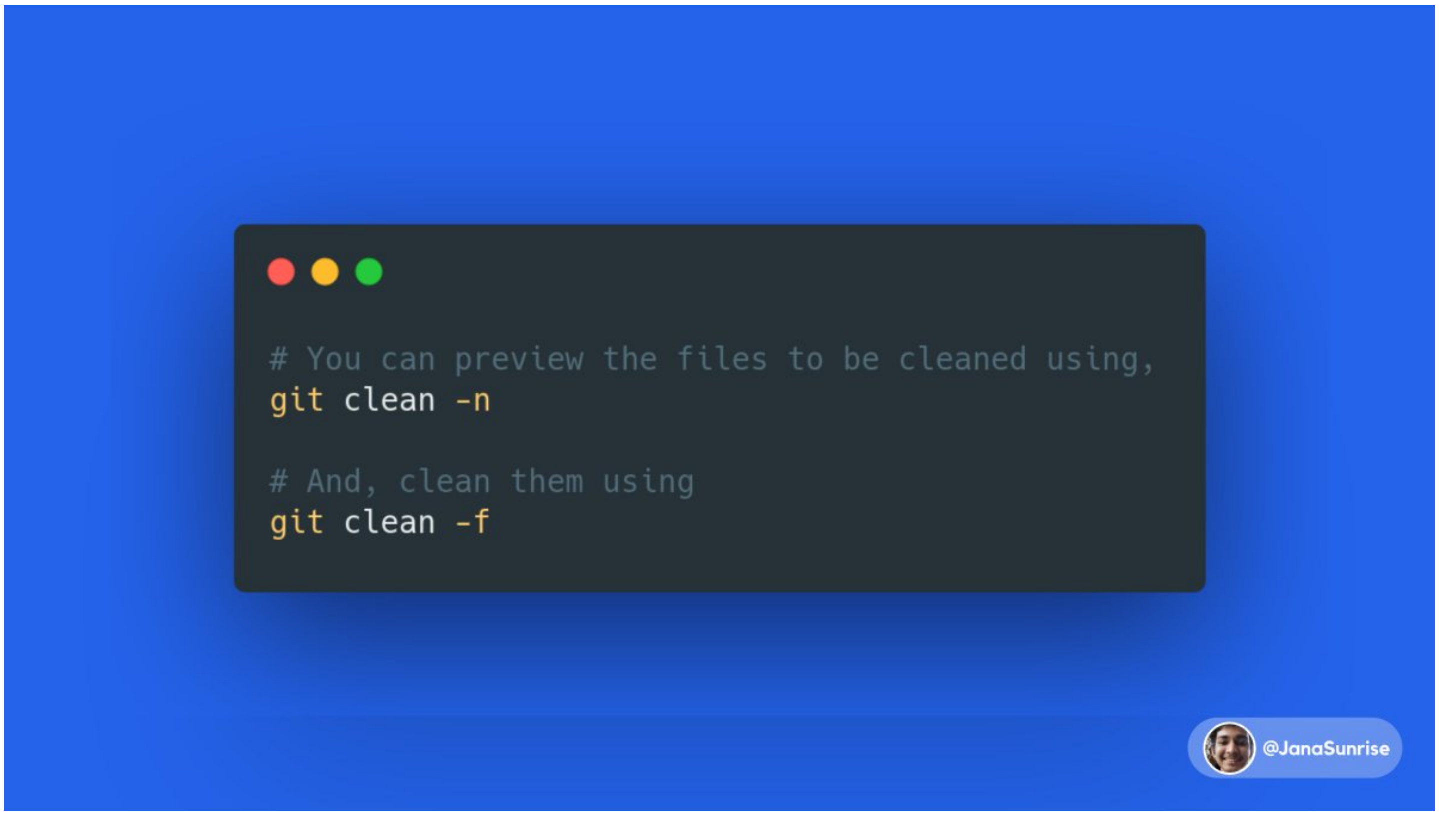


```
# You can revert back to a commit using,
git revert <commit-hash>  # You can get the hash from git log.
```

16/ Clean up unstaged files

A lot of instances, we might have the need to remove or delete all the unstaged changes (or files).

Git does provide a handy little utility for that.

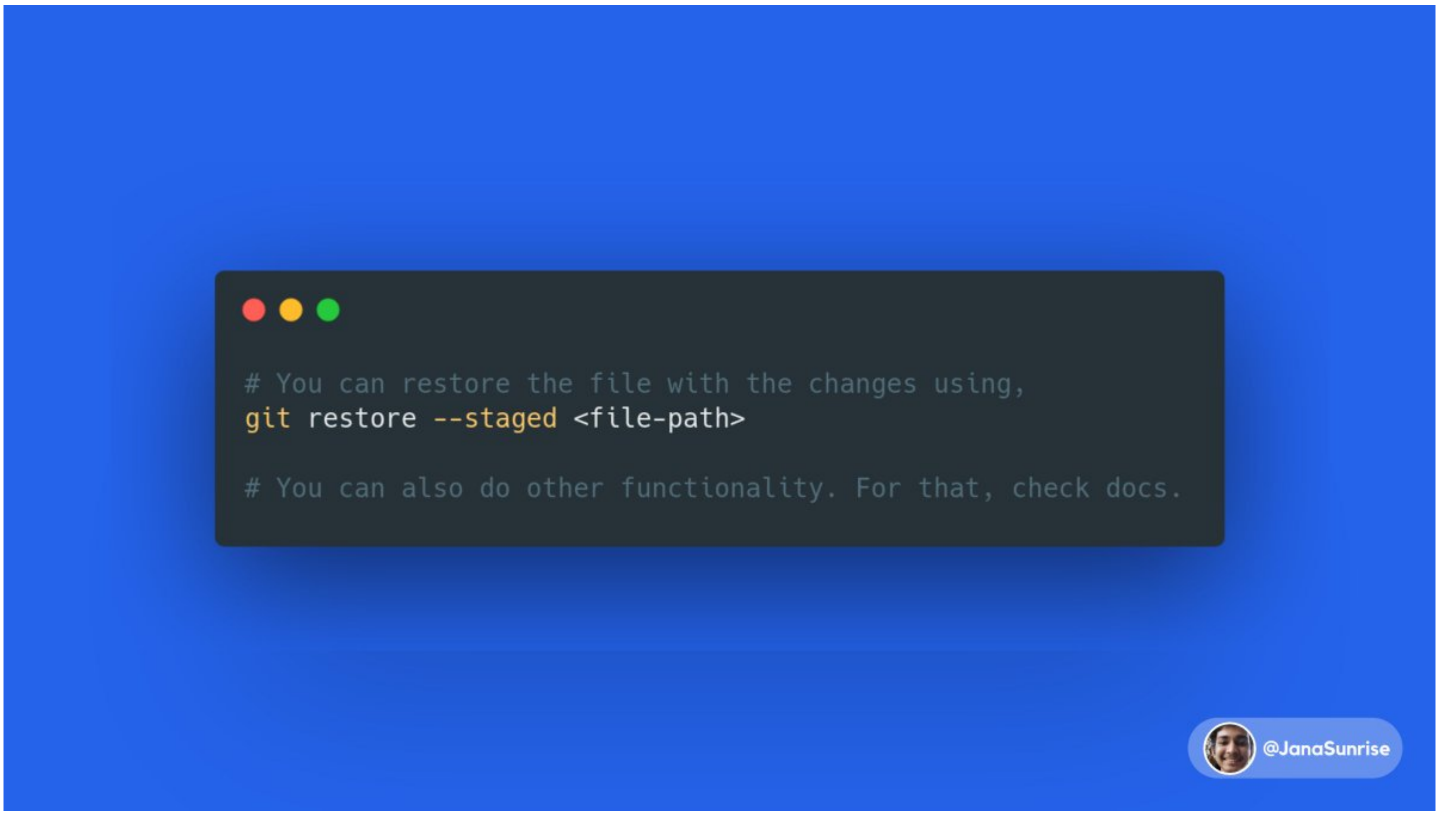


```
# You can preview the files to be cleaned using,
git clean -n

# And, clean them using
git clean -f
```

17/ Unstage a file

Let's say, we staged a file accidentally, and we meant to stage it in next commit. But how do we revert the staging?

Here is how!

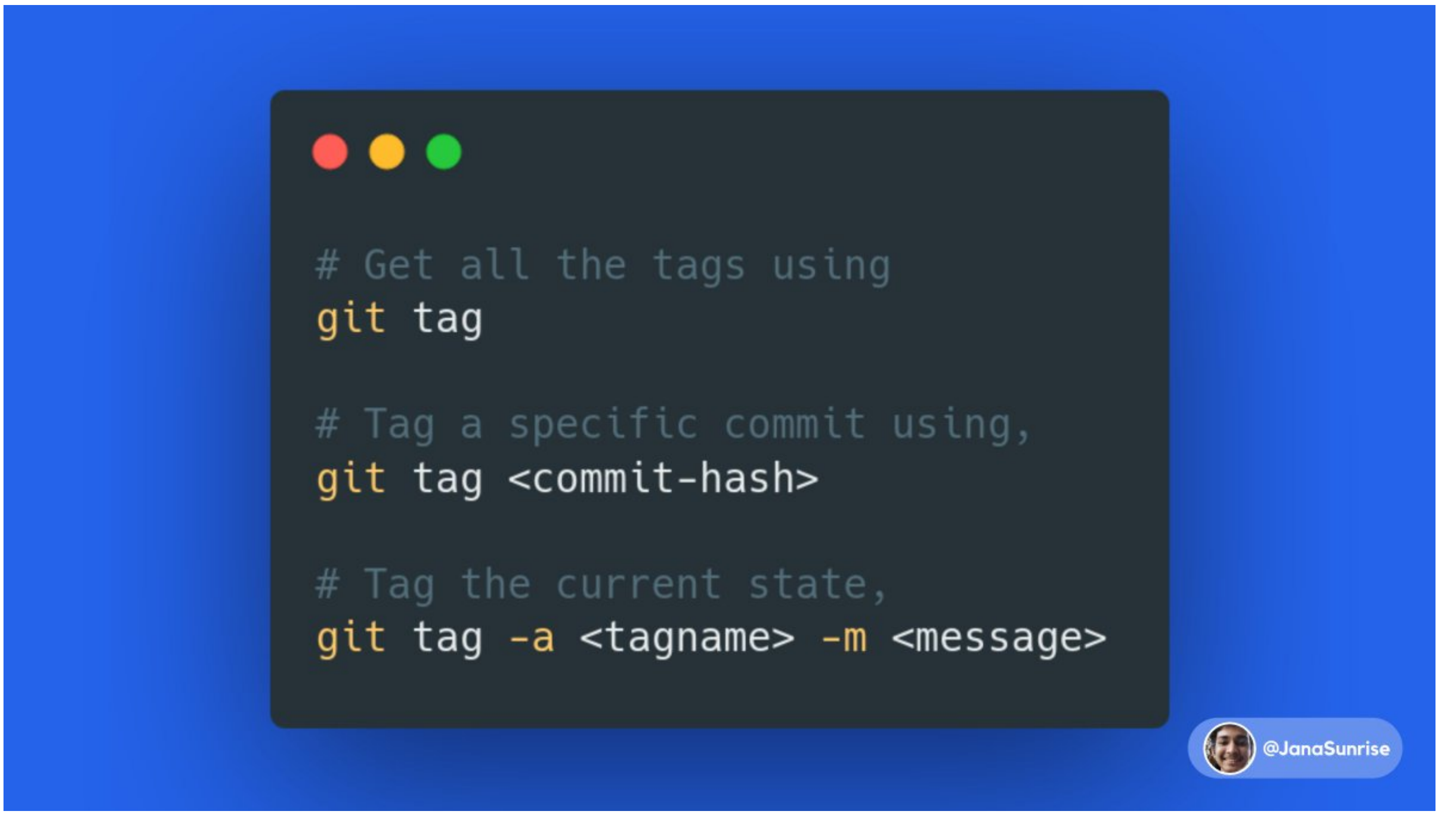


```
# You can restore the file with the changes using,
git restore --staged <file-path>

# You can also do other functionality. For that, check docs.
```

18/ Tag a specific commit

Releases of specific softwares and tools are really common. Git provides us with tags to tag the version to a specific hash, and keep it.

Really handy and awesome. Here's how,

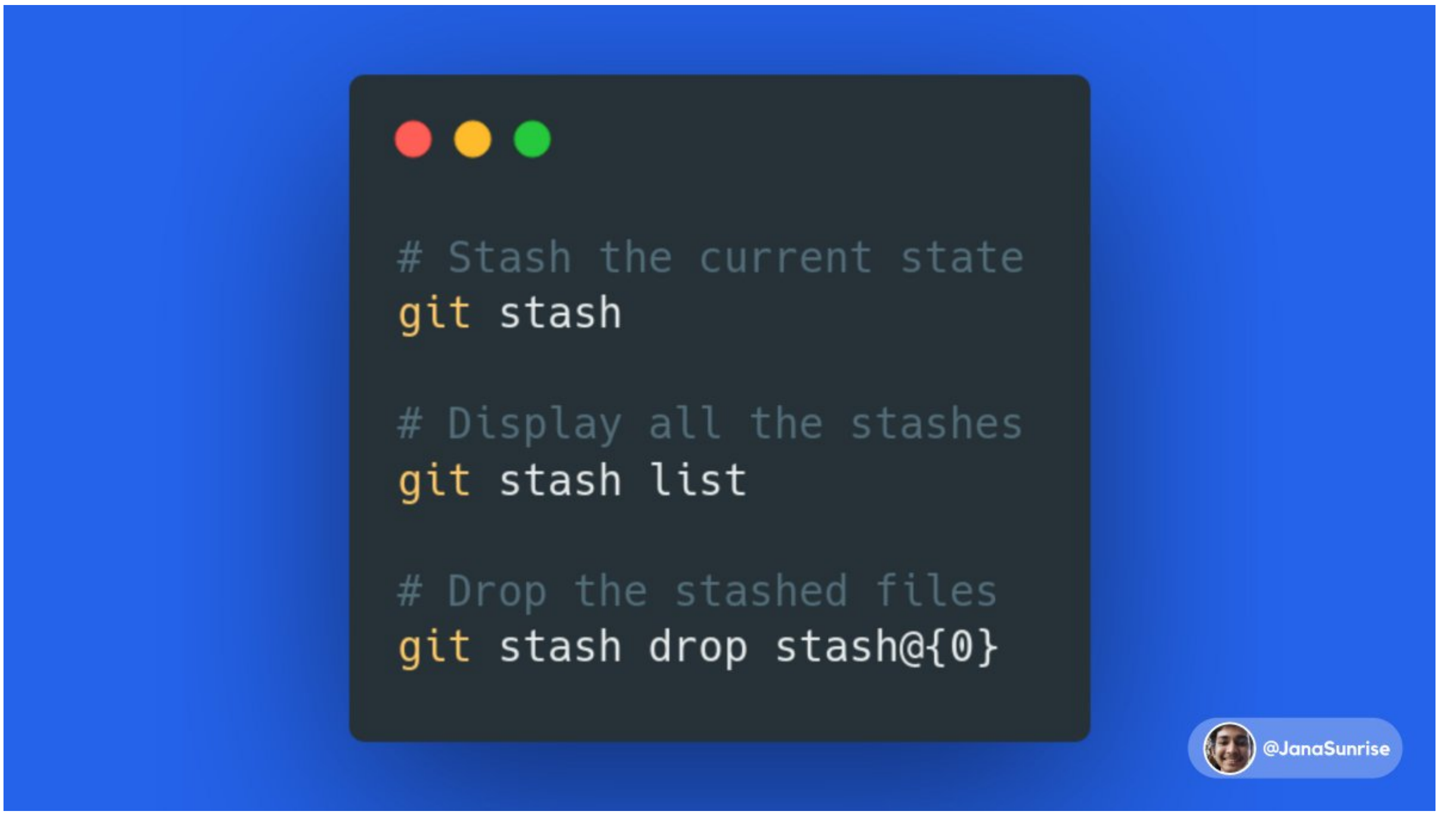


```
# Get all the tags using
git tag

# Tag a specific commit using,
git tag <commit-hash>

# Tag the current state,
git tag -a <tagname> -m <message>
```

19/ Temporarily store tracked files

Git stash allows to store tracked files temporarily for situations, such as pulling changes into my changes which are not commited yet, or some other situations.

Can be really handy! Here's how,

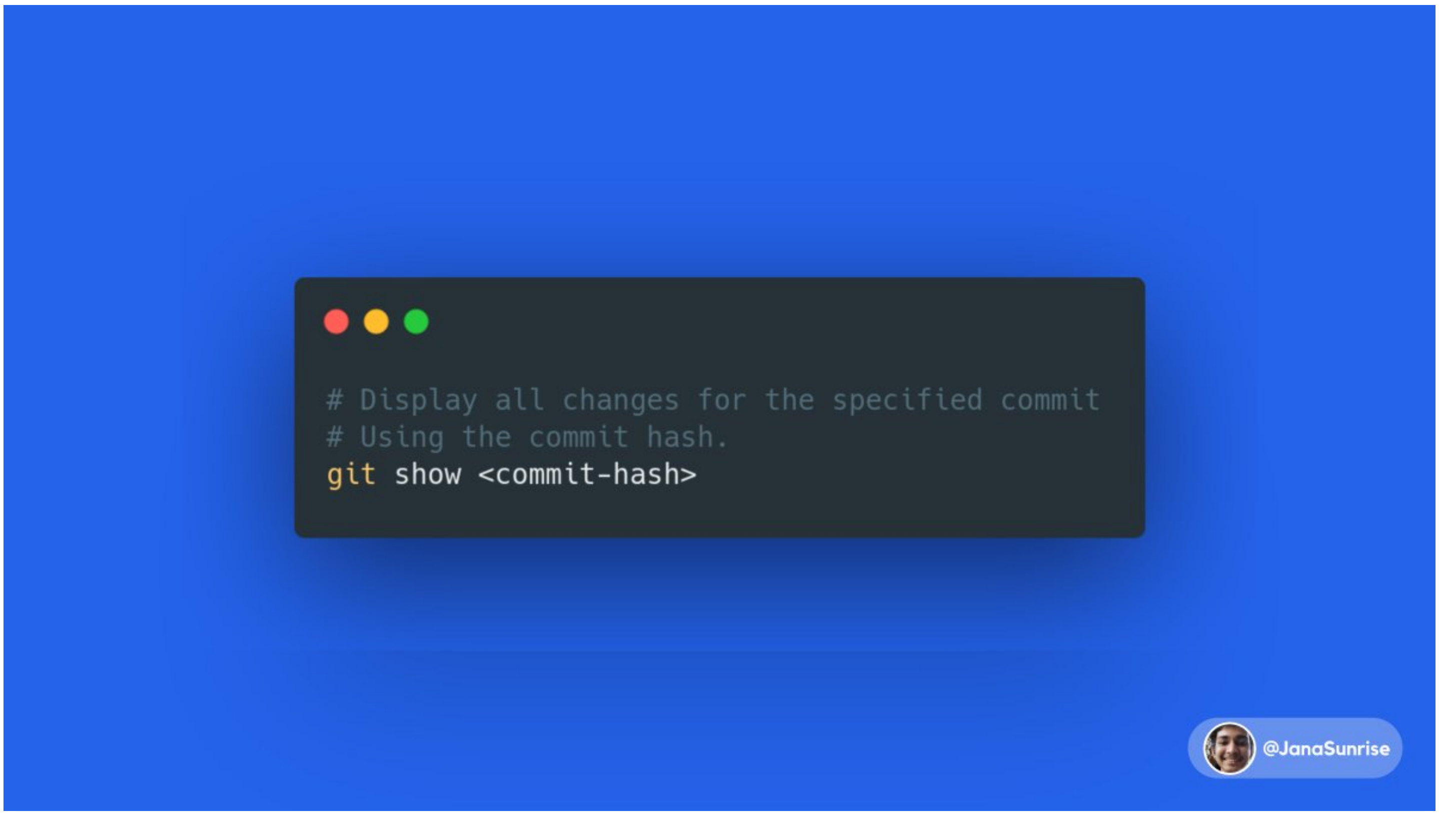


```
# Stash the current state
git stash

# Display all the stashes
git stash list

# Drop the stashed files
git stash drop stash@{0}
```

20/ Display all changes in specified commit

A lot of times, we have to evaluate the changes done in a specific commit. It's made extremely easy by git using the `show` subcommand.

Here's how,



```
# Display all changes for the specified commit
# Using the commit hash.
git show <commit-hash>
```

You have reached the end! ■

Thanks for reading! If you loved this thread,
- Follow me at @JanaSunrise ■
- Like and Retweet the first tweet ■
- And, let me know what you think! ■

I'm Sunrit, and I make content about ML, Python, Javascript and more! ■

Catch you next time ■