Psitron

Module 3

Introduction to Git

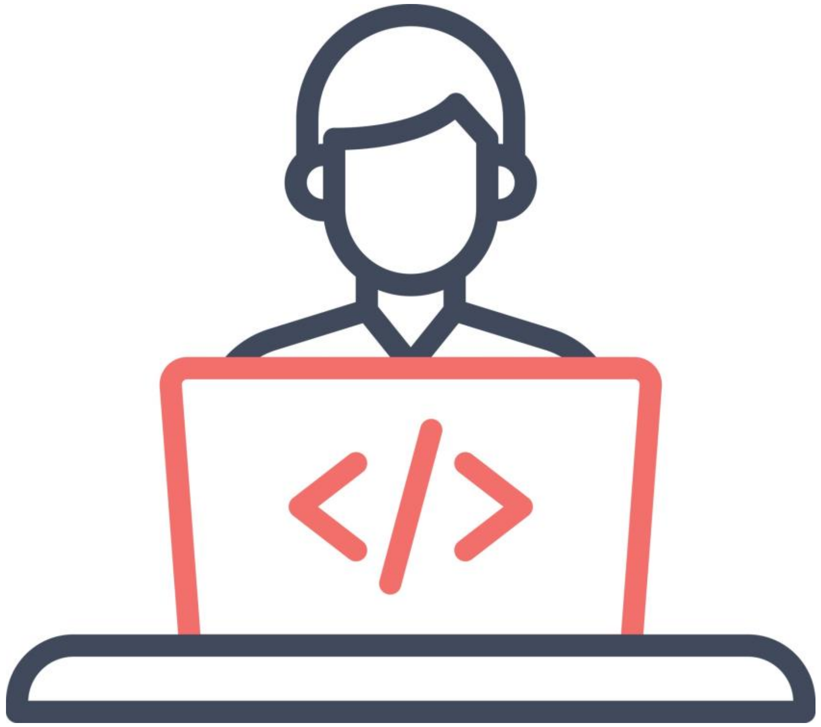
Introduction to Git

Knowledge work!

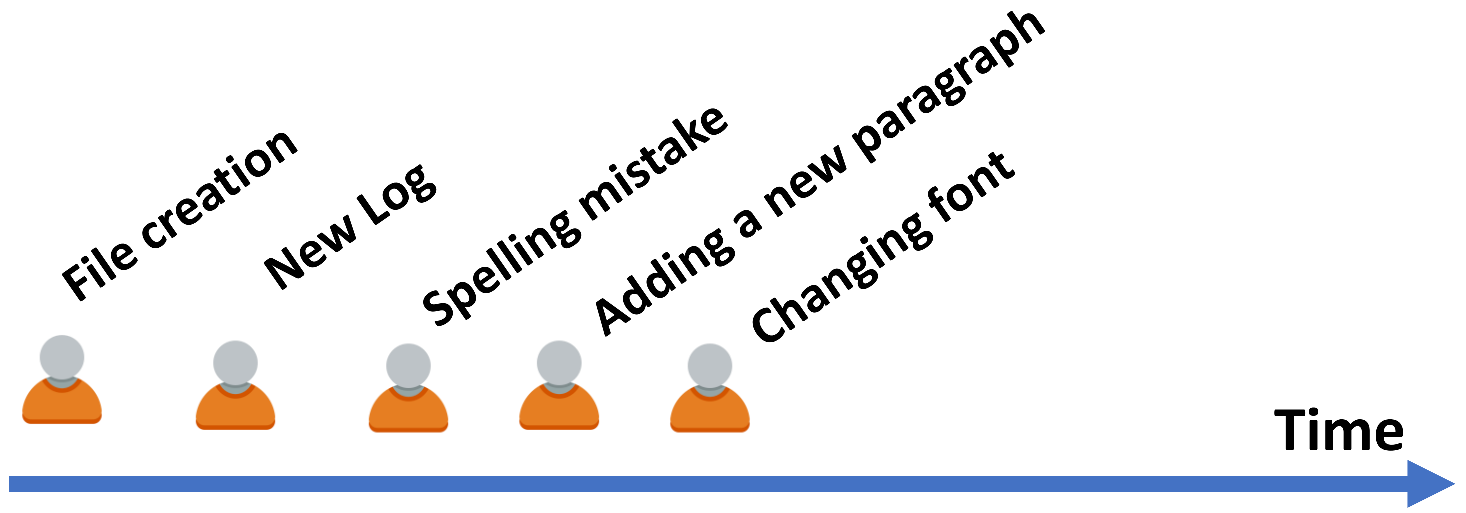
We create and edit documents (text ,code, image, etc.)

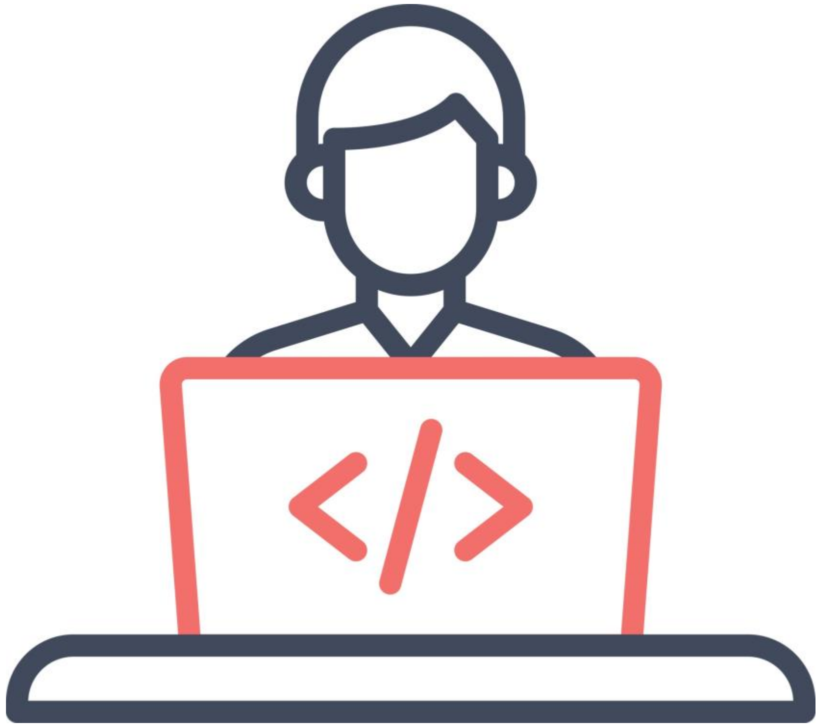
Everyday workflow

1. Create a file
2. Save it
3. Edit it
4. Save it again
5. Etc.

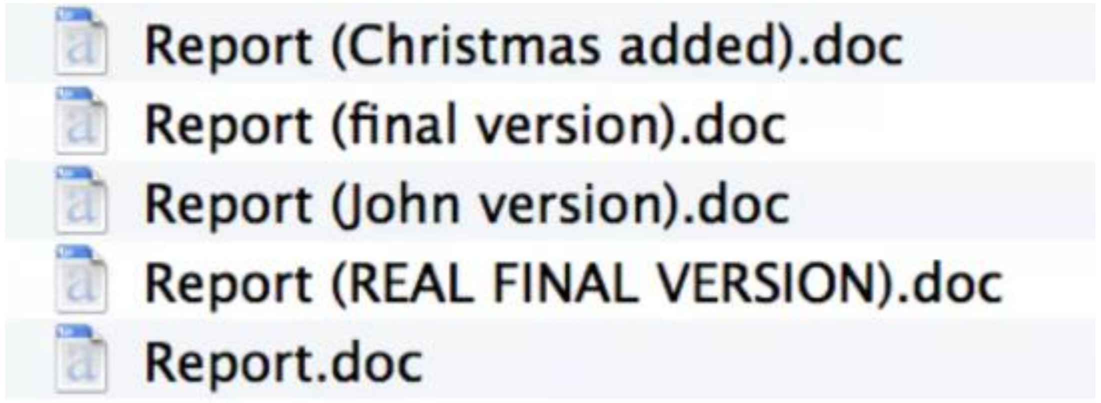


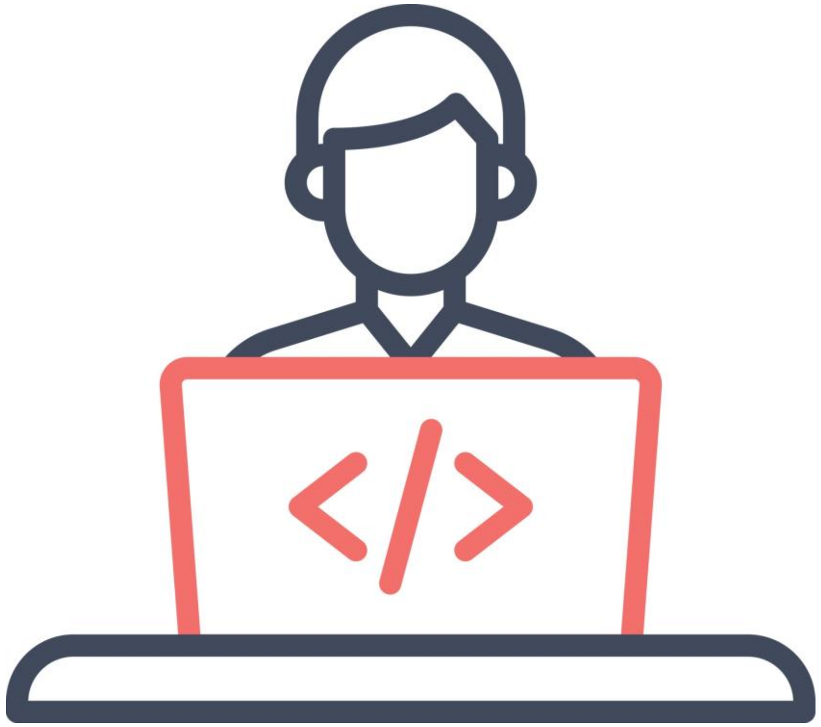
File life


File creation
New Log
Spelling mistake
Adding a new paragraph
Changing font
Time



Manual version control

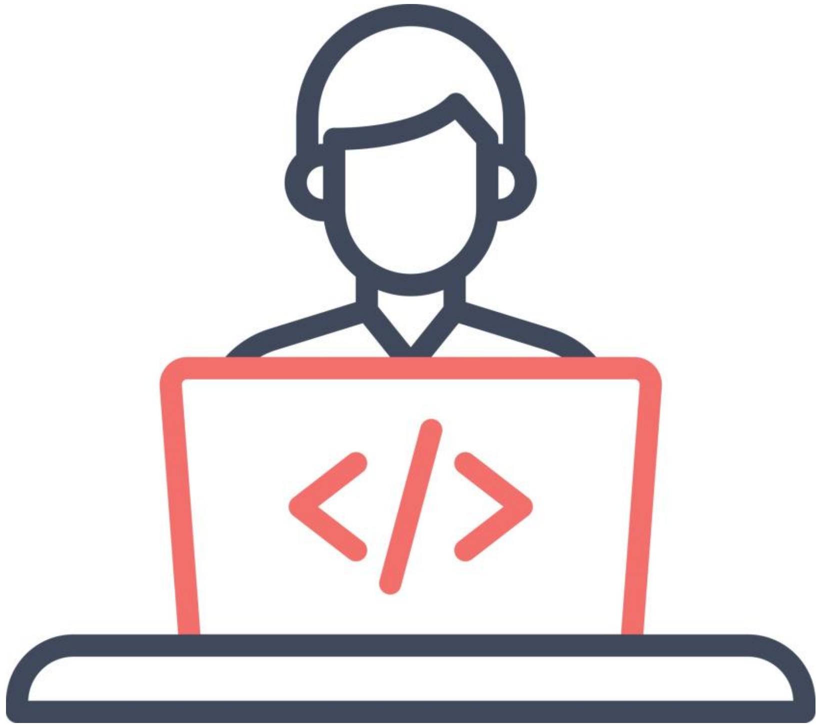

Report (Christmas added).doc
Report (final version).doc
Report (John version).doc
Report (REAL FINAL VERSION).doc
Report.doc



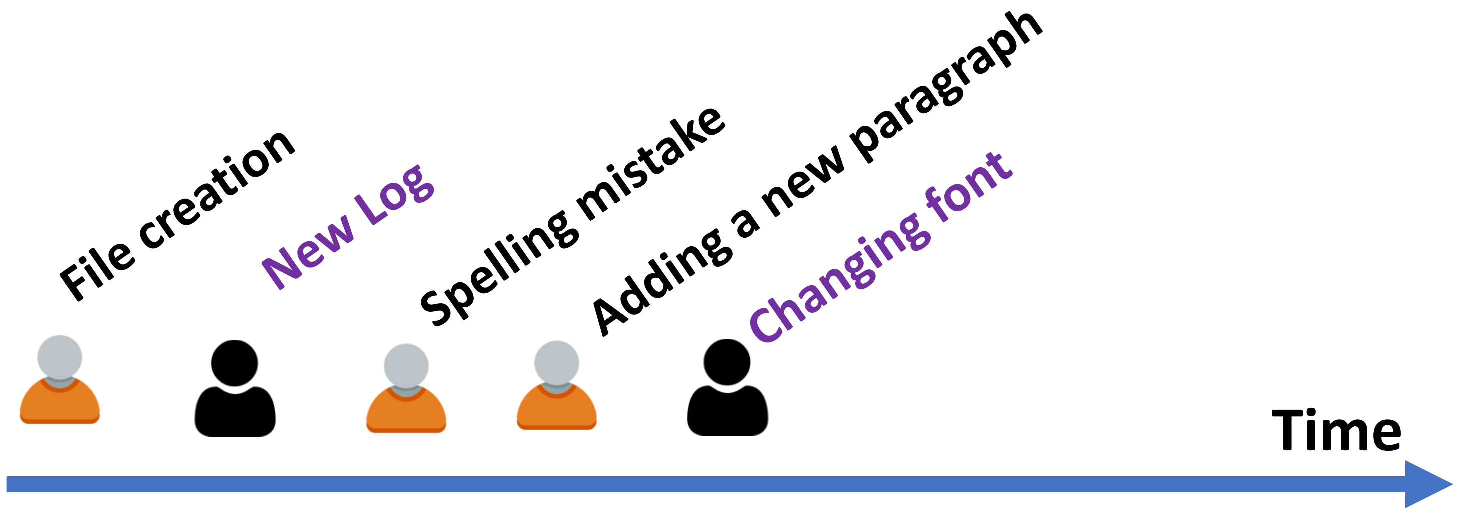
Can we automate this?

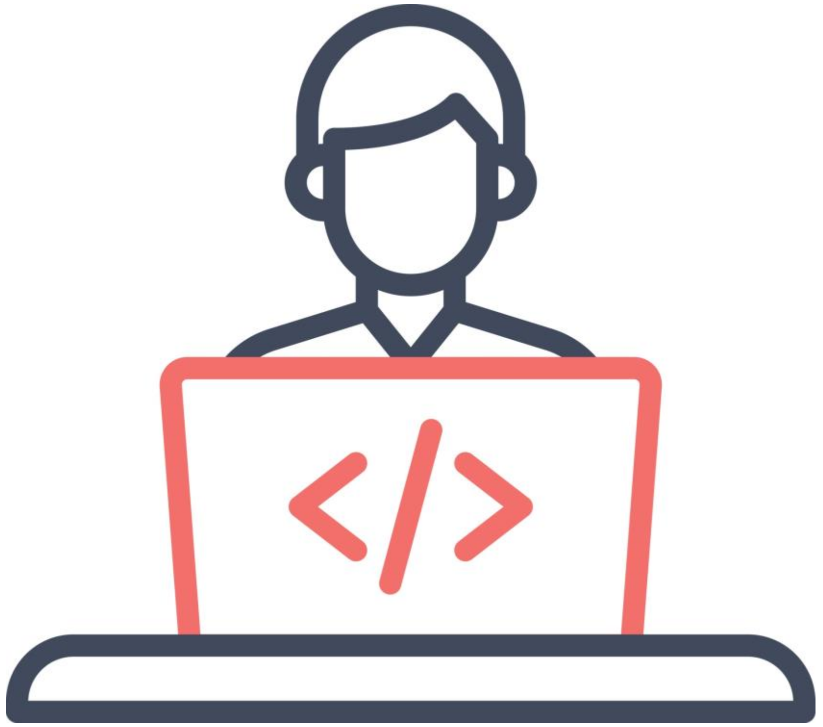
For each document version, we need to know

1. When the file was modified
2. What changes
3. Why it was modified
4. Who did the change



There's more, teams

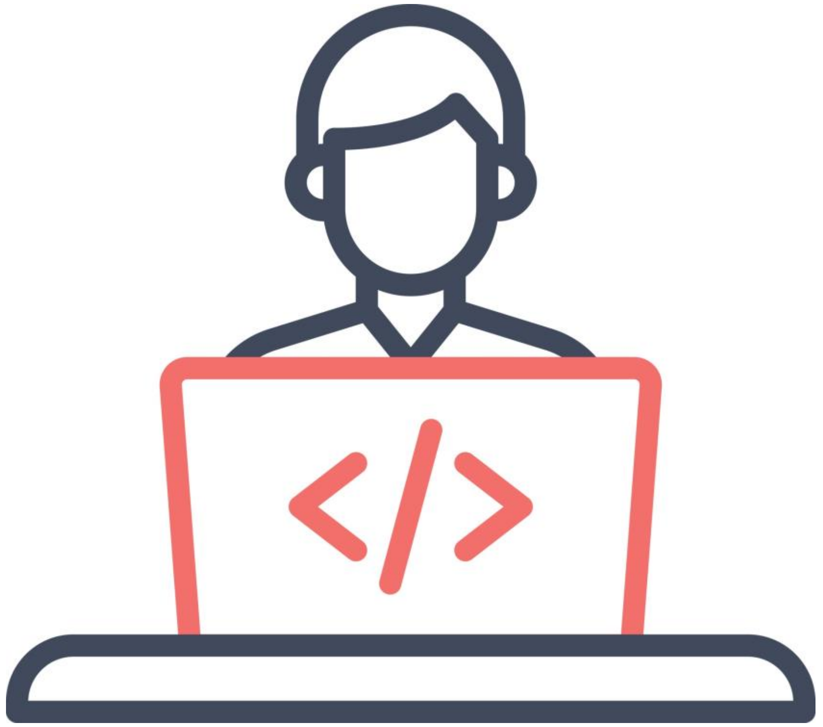




In a nutshell

We want a tool which

1. Track document version
2. Keep an history of document change
3. Foster team work



Your identity

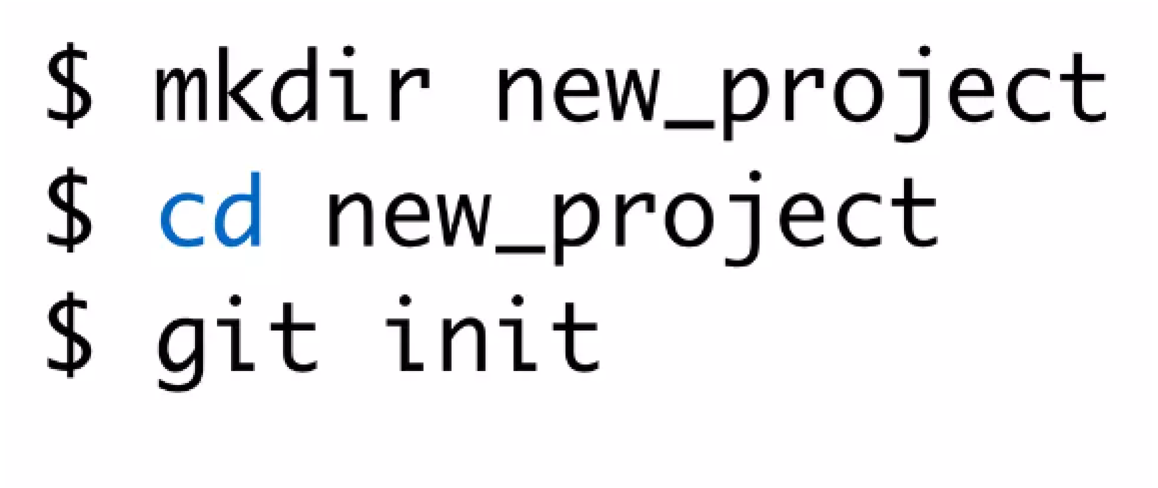
```
$ git config --global user.name "Sebastien Saunier”
$ git config --global user.email "seb@lewagon.org"
```

Psitron

Hands-on

Git

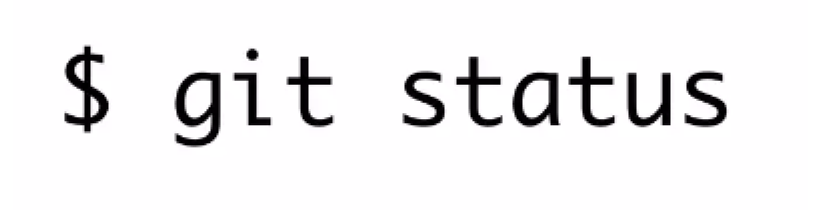
Basic commands



```
$ mkdir new_project
$ cd new_project
$ git init
```

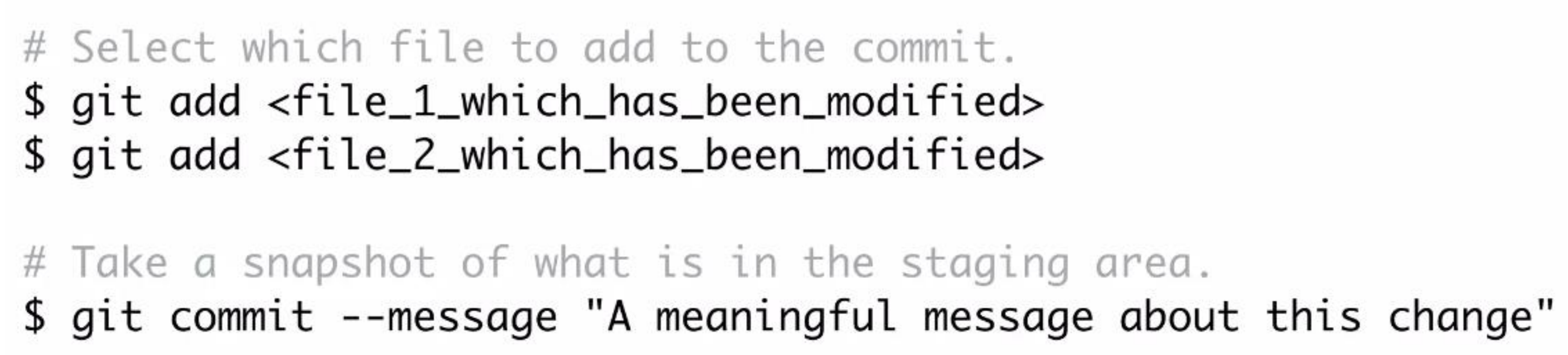
Status

Git can tell you if your folder has some modified files



```
$ git status
```

Commit

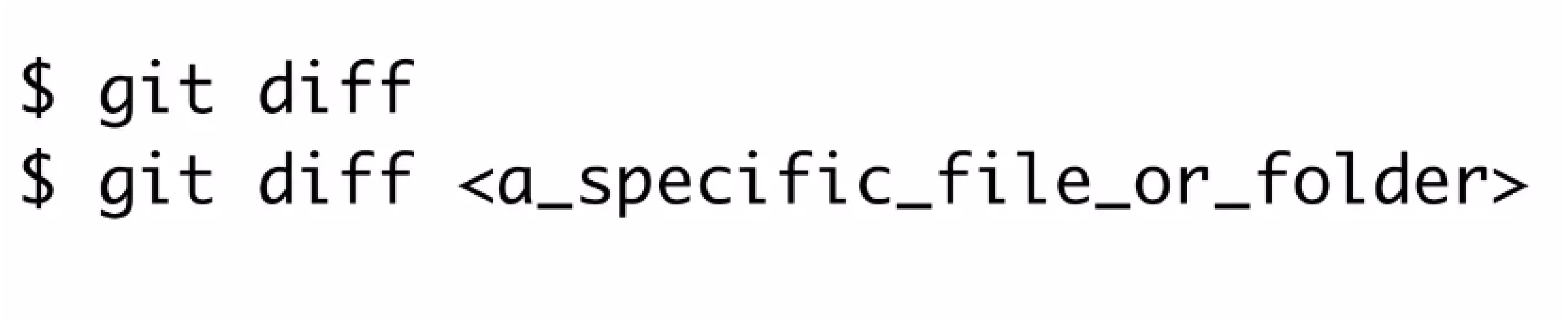


```
# Select which file to add to the commit.
$ git add <file_1_which_has_been_modified>
$ git add <file_2_which_has_been_modified>

# Take a snapshot of what is in the staging area.
$ git commit --message "A meaningful message about this change"
```

```
$ git reset --hard [commit_hash] -- [file_path]

$ git revert [commit_hash] or get revert HEAD
```

Diff

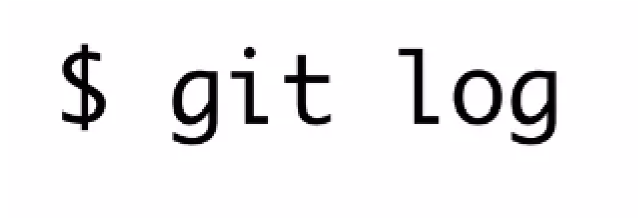
If git status tells you something changes you can inspect exactly what changes



```
$ git diff
$ git diff <a_specific_file_or_folder>
```

Log

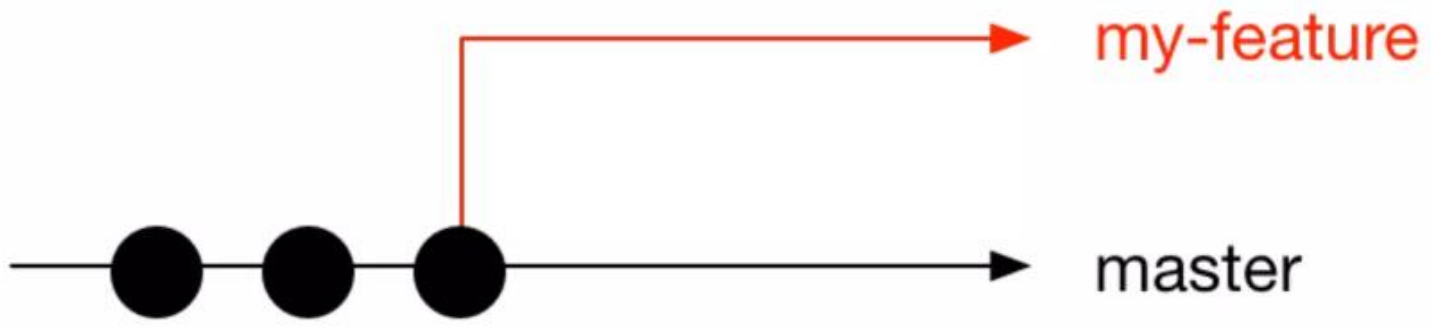
Show commit history with



```
$ git log
```

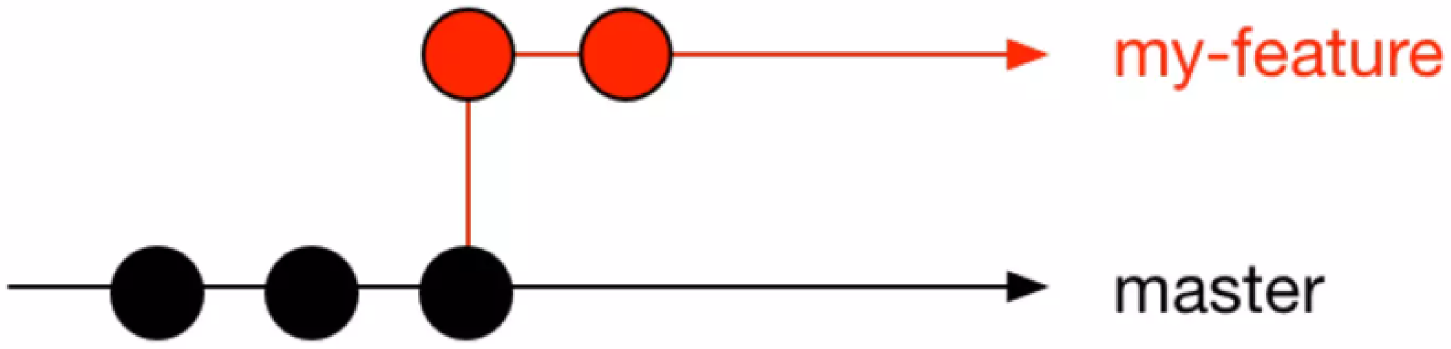
Branching

One feature = One branch



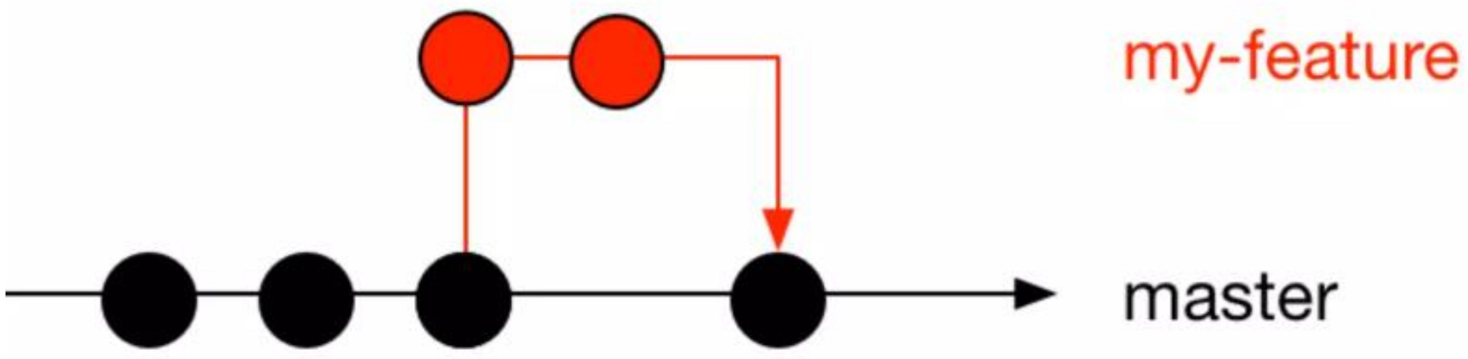
```
$ git branch my-feature
```

Working in the Branch



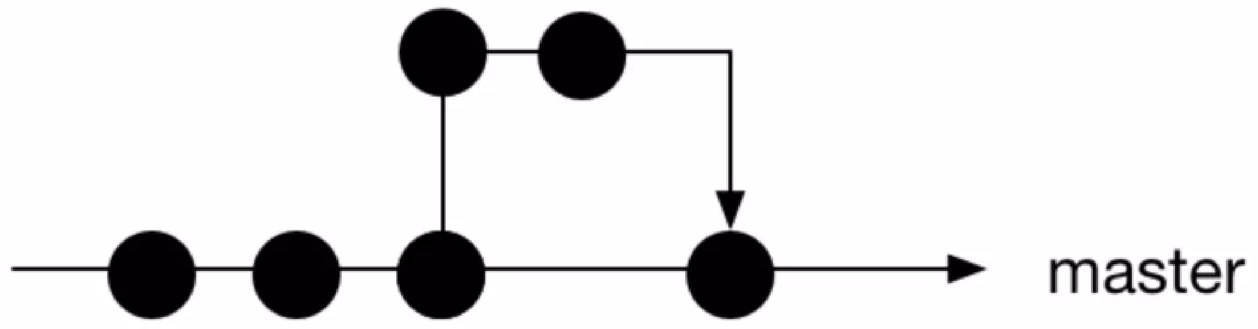
```
$ git checkout my-feature
$ git commit (x2)
```

Merge



```
$ git checkout master
$ git diff master..my-feature
$ git merge --no-ff my-feature
```

Clean Up



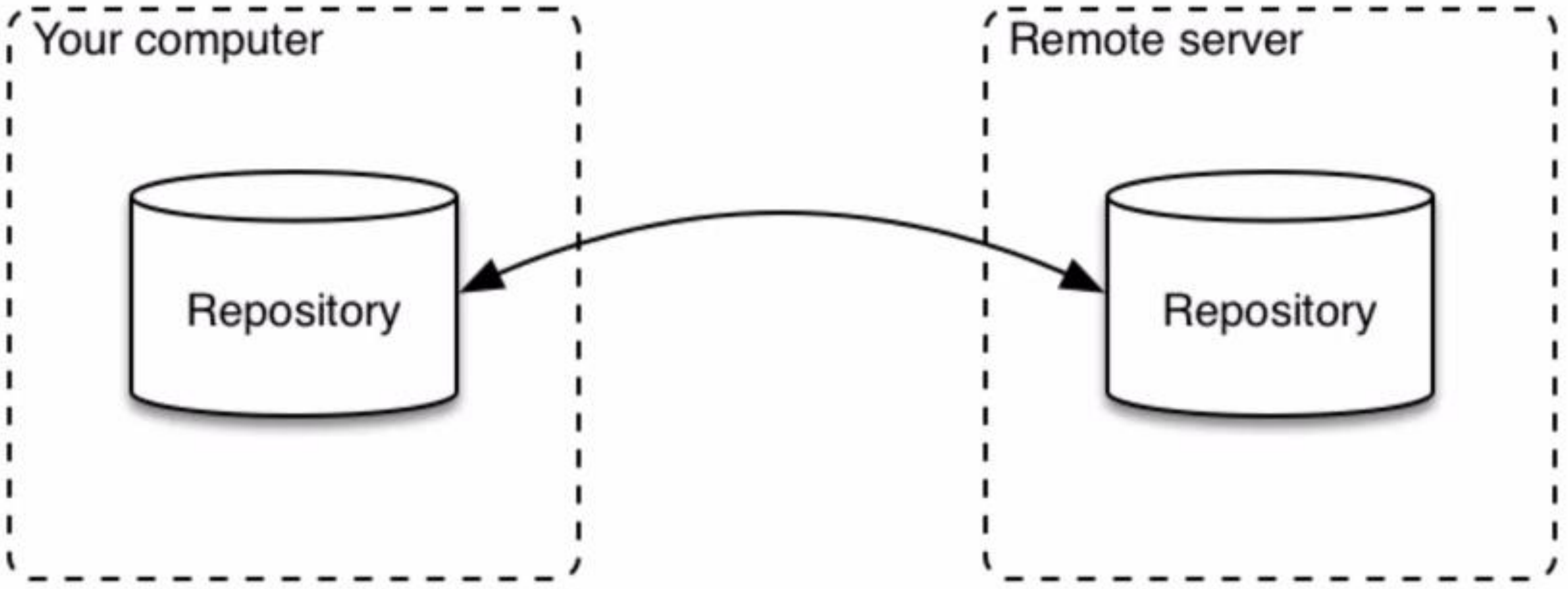
Use the following syntax to delete a local Git branch:

```
$ git branch -d my-feature
```

Use the following syntax to delete a remote Git branch:

```
$ git push [remote_project] --delete [branch_name]
```

Remote Repository



Psitron

Tools

GitHub

Google Cloud Source Repositories

ATLASSIAN Bitbucket

AWS CodeCommit

Azure Repos

Psitron

Hands-on

Create Remote Repo

We need a remote

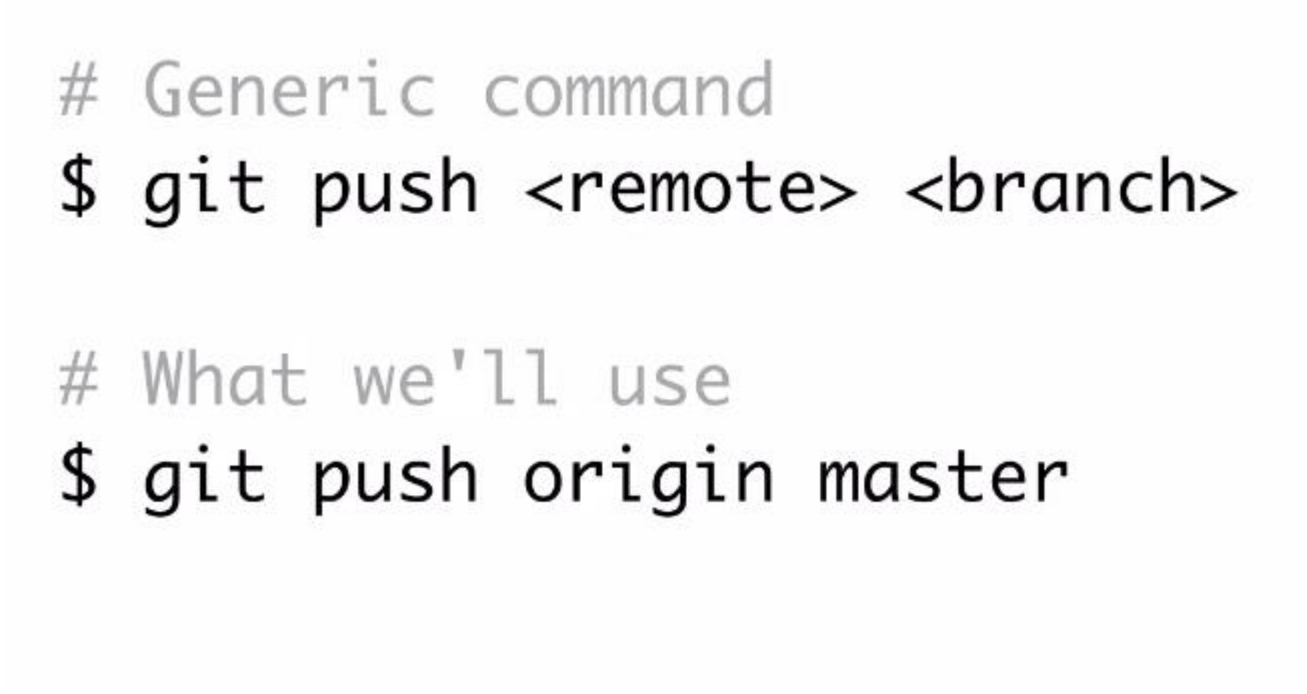
```
$ git remote add origin https://github.com/<user>/<project>.git
```

Remove a git Remote

```
$ git remote remove [remote name]
```

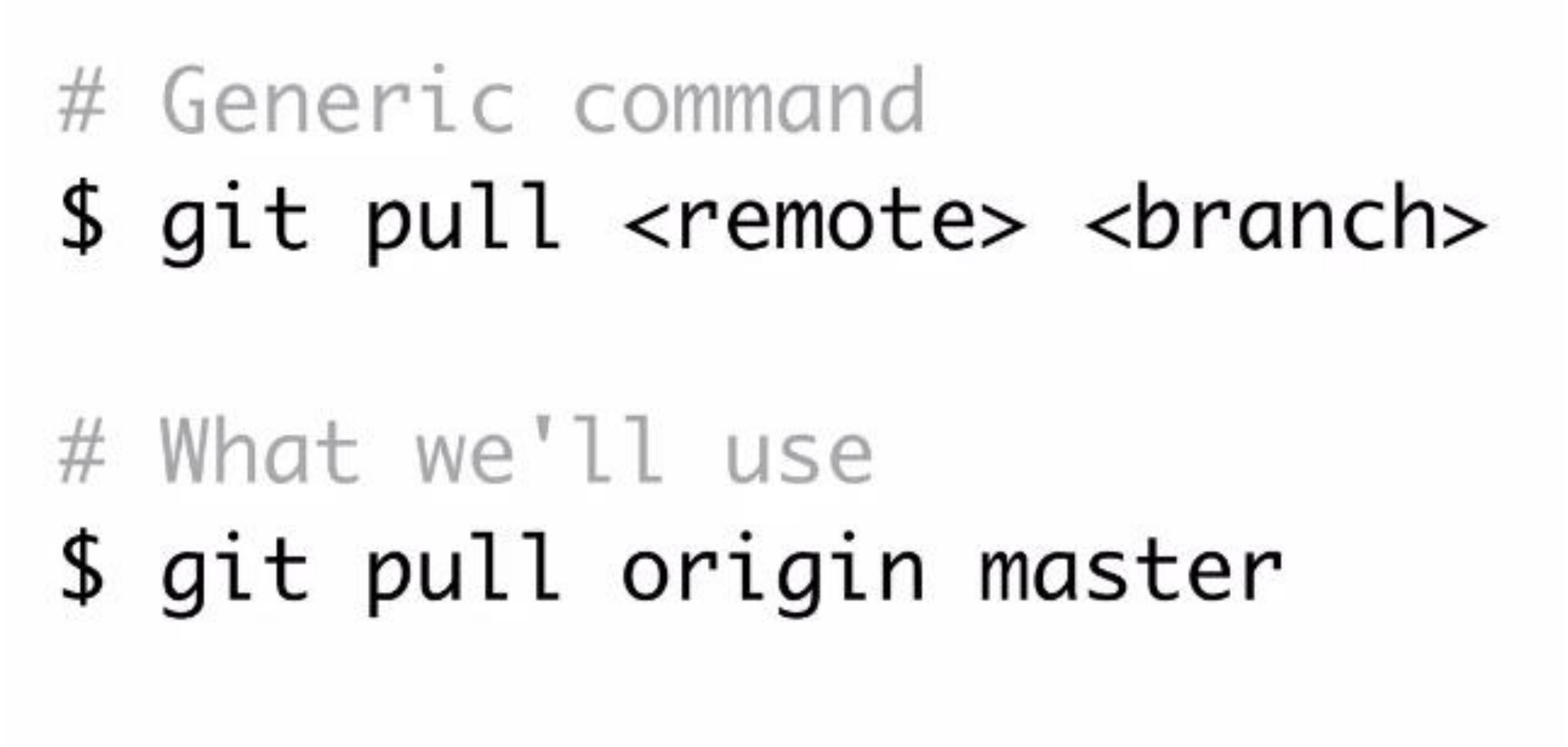
Push

Share the code with your team, and the world



```
# Generic command
$ git push <remote> <branch>

# What we'll use
$ git push origin master
```

Pull



```
# Generic command
$ git pull <remote> <branch>

# What we'll use
$ git pull origin master
```

Standard GIT branching strategies(development, feature, bug, release, UAT)

In Git, a branching strategy is a way to **organize** and **manage** different versions of your code.

- ✓ **Development branch:** This is the **main branch** where all the development work happens. It is where **developers commit their code changes**.
- ✓ **Feature branch:** A new branch is created when a developer starts working on a **new feature**, it's used to **track** and **test the feature** before merging it back to the **main development branch.**
- ✓ **Bug branch:** A new branch is created when a developer **needs to fix a bug**, it's used to **track** and **test** the fix **before merging it back** to the **main development branch.**

- ✓ **Release branch:** A new branch is created when the **development branch** is **stable**, it's used to **prepare** and **test** the **release before** it's deployed to **production.**

- ✓ **UAT branch:** A new branch is created when the **release is ready**, it's used to **test** the release with the users before it's **deployed to production.**

Understanding branching strategies

Master-feature branching

- ✓ A main branch called "master" is created, it contains the stable version of the code.
- ✓ A new branch is created from the master branch when a developer starts working on a new feature.
- ✓ The feature branch is used to track the changes and test the feature.
- ✓ The feature branch is merged back into the master branch when it's ready.

Trunk-based branching

- ✓ A main branch called "trunk" or "main" is used.
- ✓ Developers continuously commit their changes to this branch.
- ✓ This strategy is suitable for small teams or projects where development is simple and fast.