Git - Version Control

Agenda

- What is Version Control, Git, Github
- Why do we need it
- Base commands - git init, git add, git commit
- Track changes command - git status, git diff, git log
- New Github project
- Git Branching - git branch, git checkout
- Understanding CR’s and PR’s
- Resolving conflicts

digital turbine

Version Control system

Version control, also known as source control, is the practice of tracking and managing changes to software code.

Version control software keeps track of every modification to the code in a special kind of database called “**repository**”. There we can see all the file’s history, files changes, who changes them and when.

If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.

Benefits of VCS

- A complete long-term change history of every file.
- Collaboration.
- Restoring previous versions.
- Fast merging and flexible branching.
- Reliable backup copies.
- Flexibility to work offline.
- All team members can access to the projects from anywhere

Repository models

The main difference between various version control systems is the repository model - the relationship between copies of the same repository.

There are two approaches - **Centralized** and **Distributed**.

- In a **Centralized** systems the repository is maintained on a single server, with each user holding a copy of the latest state only.
- In a **Distributed** systems, each user's local copy acts as a repository and contains the complete history and metadata of all files.
    - A shared, remote repository might still exist, but isn't inherently more important than others, and is simply used for the convenience of easy access and syncing.

Centralized repository model

Pros

- Easier to understand.
- Faster initial setup - users only take what they need.
- Minimal storage needed by each user.
- Non mergeable files can be safely locked.

Cons

- Slower operations - everything requires server access.
- Cannot productively work offline.
- Single point of failure.

Distributed repository model

Pros

- Users are always productive.
- Common operations are faster.
- Each user is effectively an additional backup - no single point of failure.
- Users are independent of each other.

Cons

- Initial setup is slow as each user must download everything.
- No locking available - harder to merge non textual files.
- Much more space needed per user.

Concurrency models

Locking

- before making a change a user must obtain lock on the files they wish to work on.
- Until the lock is released, other users can view but not change the file.
- The original user releases the lock once the changes are made.

Merging

- The first user creating (or changing a file with no history) always succeeds.
- Future changes must be merged.
- Some merges can happen automatically, but complex operations must be handled manually.

GIT

- Git is a free and open source distributed version control system designed to handle everything from small to very large project with speed and efficiency
- Git was originally developed in 2005 by Linus Torvalds, the famous creator of the Linux operating system kernel.
- Git is not dependent on network access or a central server
- By far, the most widely used modern version control system in the world today is Git.
- Scalable.
- Fast.
- Easy to learn.

Github

Git is the most famous platform for hosting project using git.

It also provides extra features to make our life as a developer easier. (CI and CD tools.)

digital turbine

First steps

The first thing you should do when you install Git is to set your user name and email address.

This is important because every Git commit uses this information

$ git config –global user.name “yourName” - set your user name.

$ git config –global user.email “yourEmail” - set your email address

$ git config --global init.defaultBranch main - set main as the default branch name

$ git config --list - list all the settings Git can find.

$ git config <key> - will output specific key’s value

digital turbine

First steps

To start working with **git** we need to create a **repository** for our project.

The command to create an empty repository is

```
$ git init
```

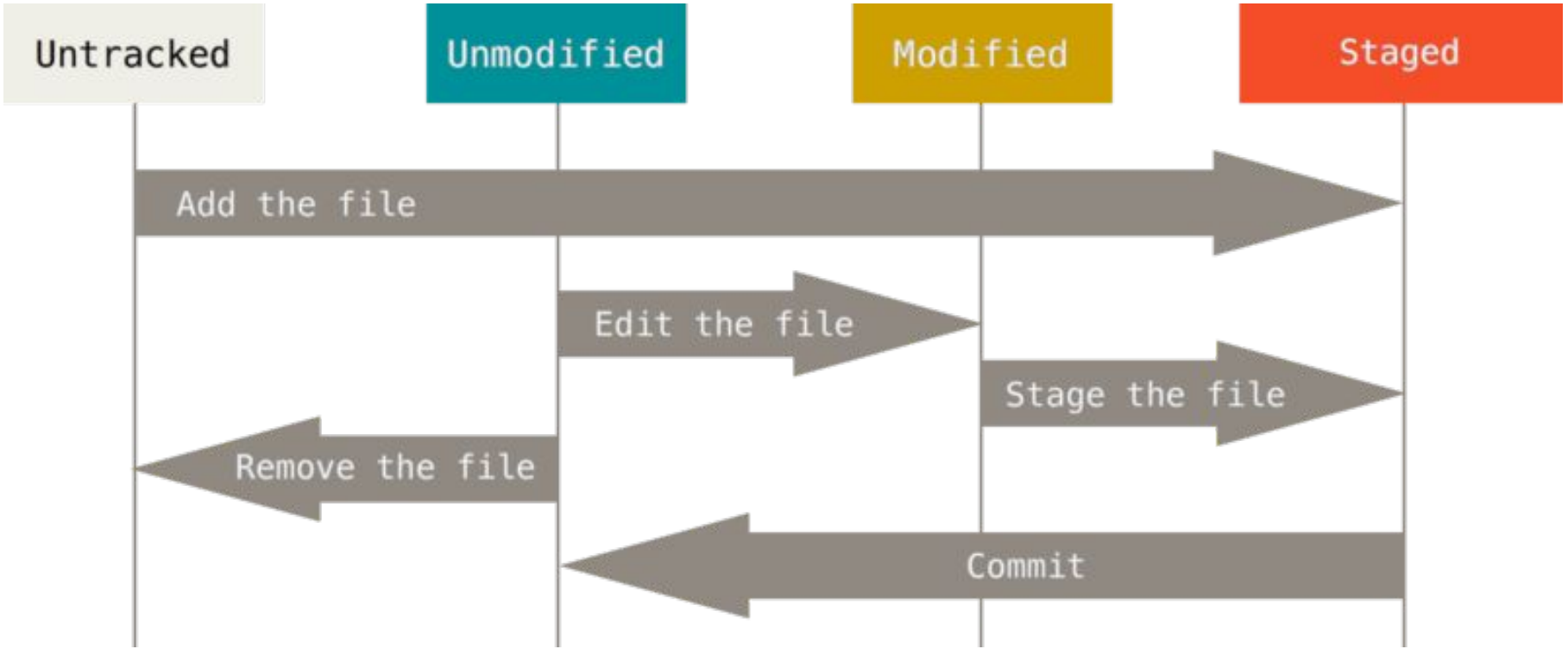
We want to run this command from our Terminal when we are in our project folder.

Git will create an empty **repository** in the folder (we can see it in the .git hidden file), and git will start tracking for changes in our project.

File states

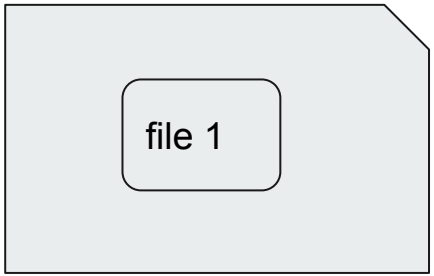
- **Untracked** - Files that git does not knows about. All your project files are in this state when you first initialize a Git repository.
- **Tracked** - Tracked files are files that were in the last snapshot, as well as any newly staged files; they can be unmodified, modified, or staged. In short, tracked files are files that Git knows about.
- **Unmodified** - Files that Git just checked them out and you haven’t edited anything.
- **Modified** - This means that the document has changed since its last committed version which is saved to our local database
- **Staged** - The file is now ready to be added to the local git database, you have marked it to go into your next commit snapshot.
- **Committed** - This state indicates that the file is safely stored in the local database.

File states

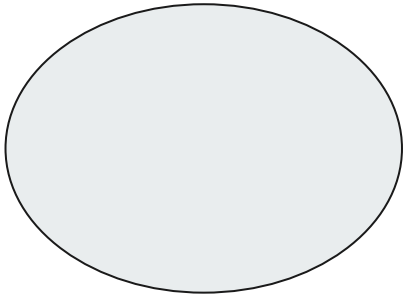


digital turbine

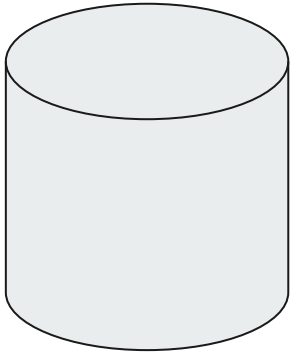
GIT Workflow



Project



Staging area

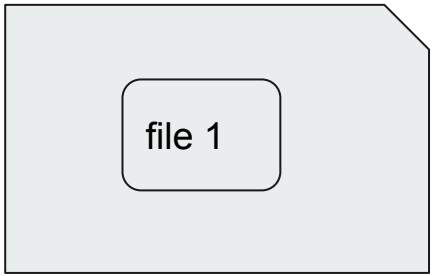


Repository

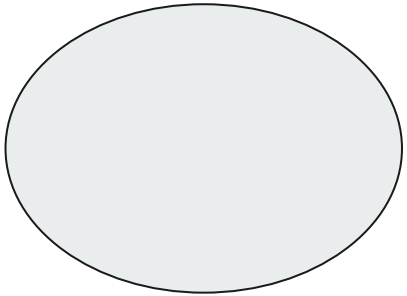
digital turbine

GIT Basic Commands

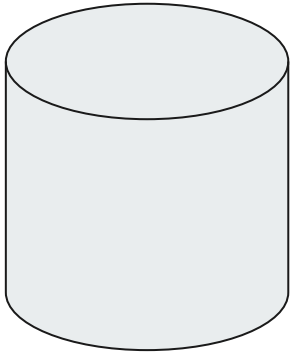
git add <filename>



Project



Staging area

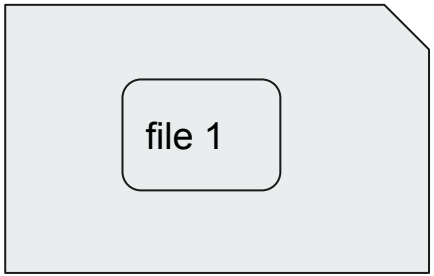


Repository

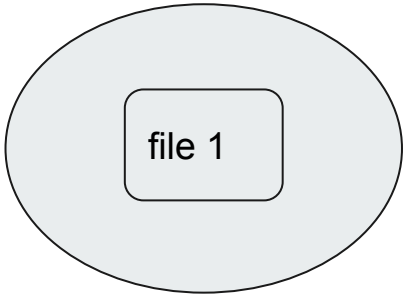
digital turbine

GIT Basic Commands

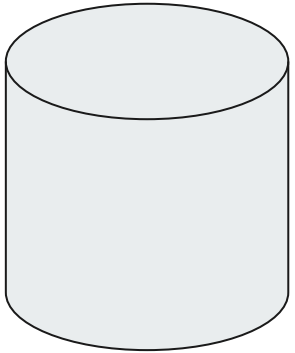
git add file 1



Project



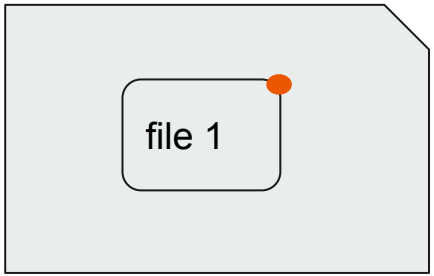
Staging area



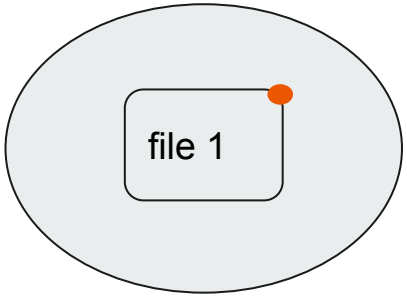
Repository

GIT Basic Commands

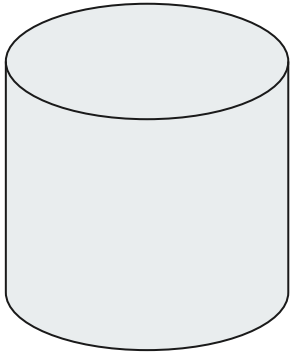
If we update the file after we added it to the staging area, git will recognize the file was modify and we will need to run `$ git add file 1` again.



Project



Staging area

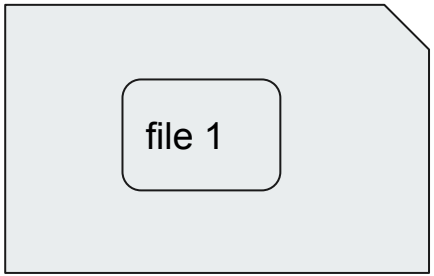


Repository

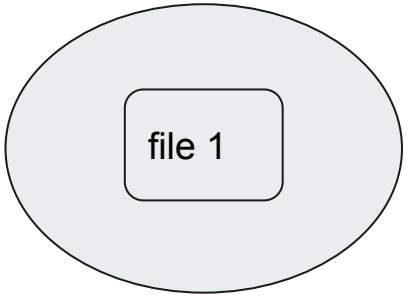
GIT Basic Commands

```
$ git add file 1
```

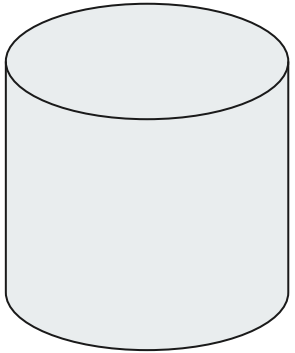
```
$ git commit -m “my message”
```



Project



Staging area

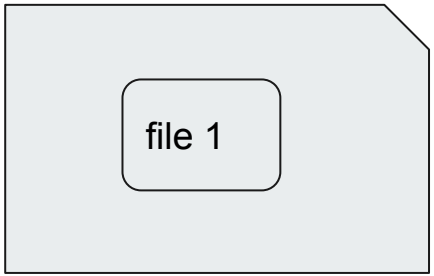


Repository

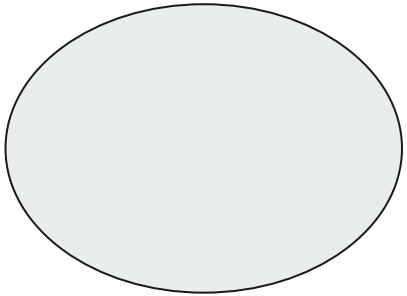
GIT Basic Commands

```
$ git add file 1
```

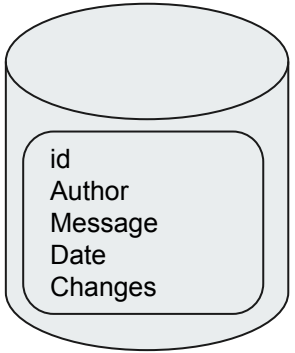
```
$ git commit -m “my message”
```



Project



Staging area



Repository

digital turbine

Track Changes

To check our git “state” we can use the command

```
$ git status
```

It let us see which changes have been staged, which haven’t and which file aren’t being tracked by git.

Track Changes

To see all our commits we can use the git command

```
$ git log
```

It will output a list with all the commit history on the repository

(to exit from the command press Q)

First Assignment

1. Create a local git repository.
2. Add you user name and email to git config
3. Add a new .txt file to the repository.
4. Add the file to the staging area.
5. Modify file - Add the text “this is file1” to the file.
6. Add the changes you made to the file.
7. Create a commit

Branches

Git branching allows developers to diverge from the production version of code to fix a bug or add a feature. Developers create branches to work with a copy of the code without modifying the existing version. You create branches to isolate your code changes, which you test before merging to the main branch

To help us with this process Git gives us the command

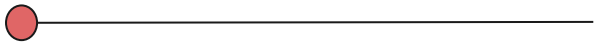
`$ git branch <branchName>`

This command lets us create a new branch that is a snapshot of the current branch we went to a branch from.

That way, we will work in an isolated environment and we can change our code however we want,

without affecting the **main** branch

Main

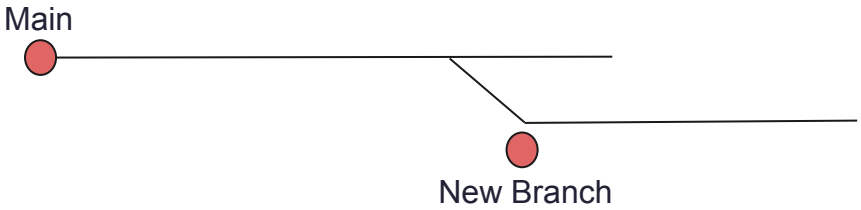


Branches

To start working on a new branch we created we need to run the command

`$ git checkout <branchName>`

The git checkout command lets you **navigate** between the branches.



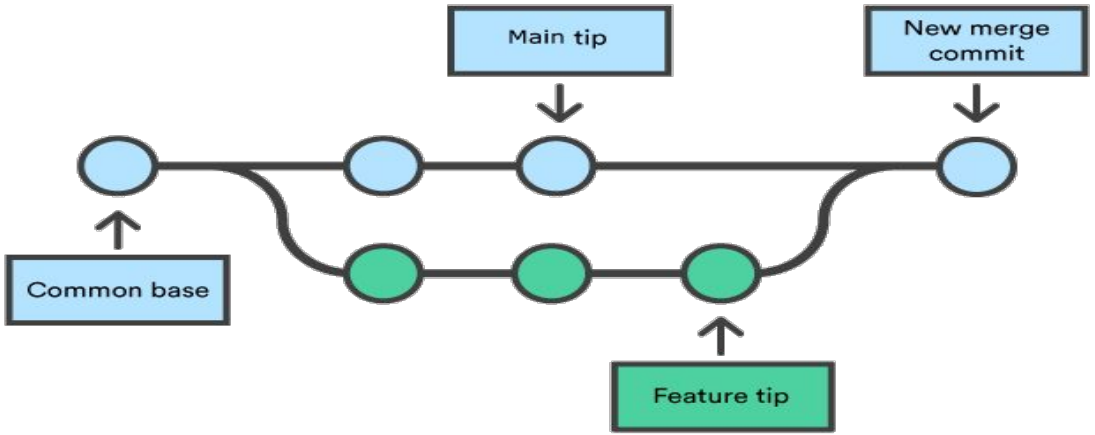
digital turbine

Branches

$ git branch - Will output all local branches
$ git branch -a - Will output all local and remote branches
$ git branch -d <branchToDelete> - Will delete a specific branch
$ git branch -D <branchToDelete> - Force delete
$ git branch -m <oldName> <newName> - Rename a local
$ git checkout <branchName> - Switch to a branch

Merge Branches

Merging is Git's way of putting a forked history back together again. The `git merge` command lets you take the independent lines of development created by `git branch` and integrate them into a single branch.



digital turbine

Second Assignment

1. From the Repo you created in the first assignment, create a new branch
2. In the new branch create a second file with some text
3. Add and commit the second file you created to the branch.
4. Merge second branch to main branch
5. Delete the branch you created.

Git Local - Remote repository

GIT have a remote repository, for users to share and sync their work.

- This remote repository is often called “**origin**”.
- Various storage solutions specializing in GIT exist, such as Github and Bitbucket, de facto acting as the remote repository.

The remote repository is mirrored in each local repository, meaning each user has a copy of the remote repository separate from their working copies.

- For example, both “**origin/master**” and “**master**” exist and can differ in content.

Merge Branches

If we want to merge branches remotely we need to open a **Pull Request.**

To open a pull request first we need to push our branch with the changes to our remote repository.

```
$ git push --set-upstream origin <branchName>
```

```
$ git push --set-upstream origin <branchName>
```

Pull Request

- A pull request is a way of notifying others about changes you've made to a git repository, and ask them to pull your code into the main development branch.
- When a pull request is made, another user (normally one of the repository admins) must view the commits and decide whether to merge or reject the request.
- Pull requests are highly important in projects with multiple contributors.
- They are supported out of the box by all major GIT hosting platforms.

Git remote commands

$git fetch - imports the latest state of branches from the remote repository, without merging into the local branches.

$git pull - runs “fetch” and then merges the remote branch into the local branch.

$git push - exports changes from the current branch to its counterpart in the remote repository.

○ Cannot be executed if one or more files have been changed in the remote repository. GIT will prompt the user to pull and resolve merge conflicts before pushing.

Third Assignment

1. Create a empty repository on your github account.
2. Set that repository as you local repository origin.
3. Push you local repository changes into your remote repository
4. Locally create a new branch
5. Add some file and modify existing file in your local repository
6. Push local branch to remote repository
7. Create Pull request to Main from you branch
8. Go over PR and merge it.

Conflicts

Git can handle most merges on its own with automatic merging features.

A conflict arises when two separate branches have made edits to the same line in a file, or when a file has been deleted in one branch but edited in the other.

Conflicts will most likely happen when working in a team environment.

Useful links and additional reading

- Learning Git Branching - Fun visual and interactive game to learn Git. ******Recommended
- Git book - basically everything you need to know about GIT.
- Comparison between different version control systems.
- Atlassian's GIT tutorial.
- Simple GIT guide with basic commands and tools.
- Github's "Hello World" tutorial.
- Git flow - a workflow and branching model that helps teams release code to production, using basic GIT concepts.

digital turbine