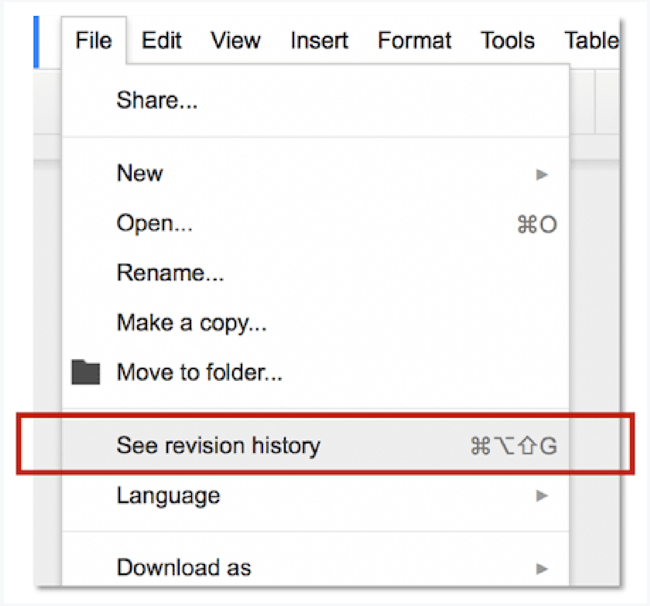
VERSION CONTROL WITH GIT

WHAT IS VERSION CONTROL?



WHAT IS VERSION CONTROL

- Developers work in (text) files
- We create many versions over time
- Hundreds of files, at certain versions, makes the system work
- We work in teams
- Each dev have their set of files

WHY DO WE NEED VERSION CONTROL AS DEVELOPERS?

- Makes sharing code and collaborating with other developers easy.
- Keeps our code tracked and safe. It tracks who, why and when the code changed.
- Makes it easy to figure out what broke your code, as you can roll back to a previous version.

DIFFERENT VERSION CONTROL SYSTEMS (VCS)

Centralized vs Distributed VCS

Centralized	Distributed
Microsoft TFVC	Mercurial
Subversion	Git
…	…
Keeps the history on the centralized server, you only download a given copy	A copy can be available on a centralized server (ie. GitHub, GitLab), but you have the entire history locally

WHY GIT?

- Lots of learning resources are publicly available.
- Does not require you to be connected to the internet to use.
- Very secure. Ensures that the history is fully traceable.
- By far the most popular VCS today.

git

!=

GitHub

WHAT IS GITHUB?

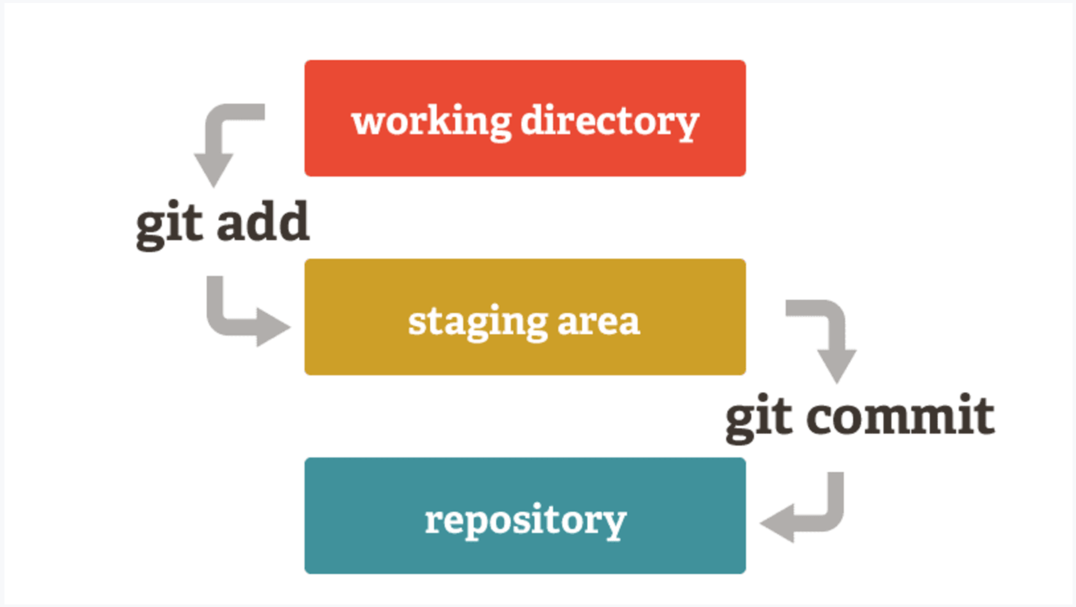
Web based hosting service for our code repositories.

Alternatives to Github are GitLab and Bitbucket.

TERMINOLOGY

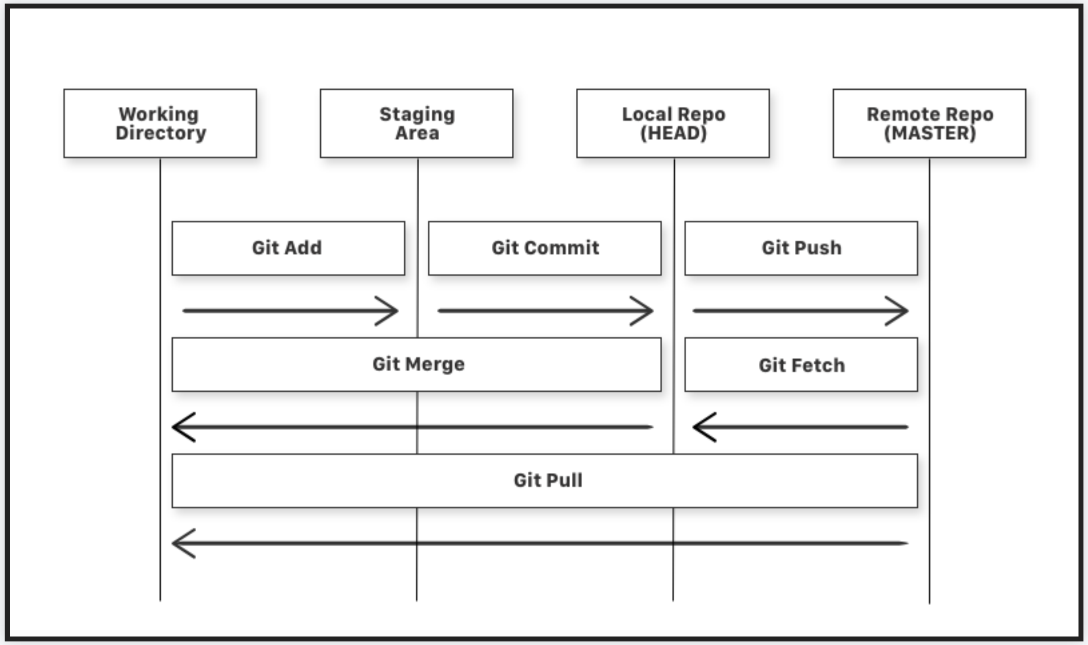
Name	Description
Repository	A location where code is stored, either on your computer or somewhere else. Also called a ‘repo’.
Remote Repository	Repositories that are hosted on the Internet, for example on Github.
Clone	Copy a repository and all its history so that you can work on it on your local machine.
Staging	Prepare one or multiple files for a commit.

HOW DOES GIT WORK



Git Staging Area: Explained Like I'm 5

GIT WORKFLOW



COMMITING FILES TO A LOCAL GIT REPOSITORY

Git Command	Description
`git init`	Makes your local directory a git repository.
`git status`	Shows the state of the local working directory and the Staging area.
`git add <fileName>`	Adds the specific file in the local repository and stages it for commit.
`git add .`	Adds all the files in the local repository and stages them for commit.
`git commit -m "message"`	Applying any changes you have staged to the local repo. Write a commit message in present tense.

BRANCHES

- Different features of the code base can live in their own separate branches.
- We often have one branch from which the other branches originate.
- In Git, this branch is called ~~master~~ **main**.
- When we use a centralized server, it's often called **origin/main**.
- Branches have to be *merged* back onto main.

BRANCHES

Git Command	Description
`git branch <branchname>`	Create a new local branch based on your current branch.
`git checkout <branchname>`	Switch to the new local branch.
`git checkout -b <branchname>`	Create a new local branch and immediately switch to it.
`git branch -d <branchname>`	Removes the local branch.

COLLABORATING

Git Command	Description
`git clone <url>`	Create a remote connection called origin pointing back to the cloned repository.
`git remote -v`	List any remote connections you have to other repositories.
`git remote add <url>`	Create a new connection to an existing remote repository.
`git push <remotename> <branchname>`	Apply your commited changes to the specified branch of the remote repository.
`git fetch`	Get any new changes made to the remote repository (all branches or specify one).
`git merge <remotename>/<branchname>`	Synchronize the current local branch with the main branch on the origin remote repo.
`git pull`	Git fetch and immediately merge.

MERGING

To join two or more changes of the same repository

May cause a conflict!

CONFLICTS

Let's create one and then solve it...

CREATE A NEW FILE

Create a new file on main

```
function iceCream() {
  return 'I like ice cream!';
}

console.log(iceCream());
```

Commit the file to our repository

```
$ git add .
$ git commit -m "print ice cream message"
```

```
$ git log
```

```
commit f565a05264570544b9fb91104d012d8d5b582e85 (HEAD -> main)
Author: Levy Fekete <levy@salt.dev>
Date:   Wed Feb 26 15:00:43 2020 +0200

    print ice cream message
(END)
```

ADD A NEW FEATURE

Create a new feature branch

```
$ git checkout -b feature
```

Or

```
$ git branch feature
$ git checkout feature
```

Edit the file

```
function iceCream(taste) {
  return `I like ${taste} ice cream!`;
}

console.log(iceCream('vanilla'));
```

Look at the diff

```
$ git diff
```

```
diff --git a/index.js b/index.js
index d351805..f5bd868 100644
--- a/index.js
+++ b/index.js
@@ -1,5 +1,5 @@
-function iceCream() {
-  return 'I like ice cream!';
+function iceCream(taste) {
+  return `I like ${taste} ice cream!`;
}

-console.log(iceCream());
+console.log(iceCream('vanilla'));
(END)
```

Commit the file

```
$ git add .
$ git commit -m "update function and prefer vanilla ice cream"
```

CURRENT STATE

```
// main Branch
function iceCream() {
  return 'I like ice cream!';
}

console.log(iceCream());
```

```
// Feature Branch
function iceCream(taste) {
  return `I like ${taste} ice cream!`;
}

console.log(iceCream('vanilla'));
```

CREATE THE CONFLICT

Switch back to main

```
$ git checkout main
```

Edit the file in a way that it conflicts with the feature branch

```
function iceCream() {
  return 'I like chocolate ice cream!';
}

console.log(iceCream());
```

Commit the file

```
$ git add .
$ git commit -m "prefer chocolate flavour"
```

CURRENT STATE

```
// main Branch
function iceCream() {
  return 'I like chocolate ice cream!';
}

console.log(iceCream());
```

```
// Feature Branch
function iceCream(taste) {
  return `I like ${taste} ice cream!`;
}

console.log(iceCream('vanilla'));
```

REALIZE THE CONFLICT

Merge feature into main

```
$ git merge feature
```

Git says

```
Auto-merging index.js
CONFLICT (content): Merge conflict in index.js
Automatic merge failed; fix conflicts and then commit the result.
```

STAY CALM!

THE FILE NOW CONTAINS BOTH VERSIONS

```
<<<<<<< HEAD
function iceCream() {
  return 'I like chocolate ice cream!';
=======
function iceCream(taste) {
  return `I like ${taste} ice cream!`;
>>>>>>> vanilla taste
}
console.log(iceCream('vanilla'));
```

WE NOW HAVE TO DECIDE HOW TO SOLVE THIS CONFLICT

The options are:

- Discard our changes
- Keep our changes and discard the changes made in main
- Make some kind of intelligent decision

SOLVE THE CONFLICT

Edit the file to support both versions

```
function iceCream(taste = 'chocolate') {
  return `I like ${taste} ice cream!`;
}

console.log(iceCream('vanilla'));
```

Check status of files

```
$ git status
```

GIT SAYS:

```
On branch main
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

  both modified:   index.js
```

COMMIT THE MERGE

```
$ git add .
```

```
$ git commit -m "fix merge conflict"
```

CONFLICT SOLVED! ✔

```
$ git status
On branch main
nothing to commit, working tree clean
```

.GITIGNORE

Add files here you don't want git to track

```
node_modules
dist
*.log
.DS_Store
.env
config.local.json
password.txt
```

```
npx gitignore node # creates a good gitignore file
```

Some common operations

Command	Description
`git help`	Get help!
`git log`	Show the commit log on the current branch
`git commit --amend`	Change the latest commit message
`git stash`	Move the current changes into a stash
`git stash pop`	Apply the stash onto the current branch
`git reset`	Discard all changes added to stage

SOME USEFUL LINKS:

Atlassian Git Tutorial https://www.atlassian.com/git/tutorials

Github Git Tutorial https://try.github.io/

Learn the basics of Git in under 10 minutes https://www.freecodecamp.org/news/learn-the-basics-of-git-in-under-10-minutes-da548267cc91/

Advanced Git with Keith Dalby Git more done

Cure git confusion