GIT & GITHUB

INTRO TO THE COMMAND LINE

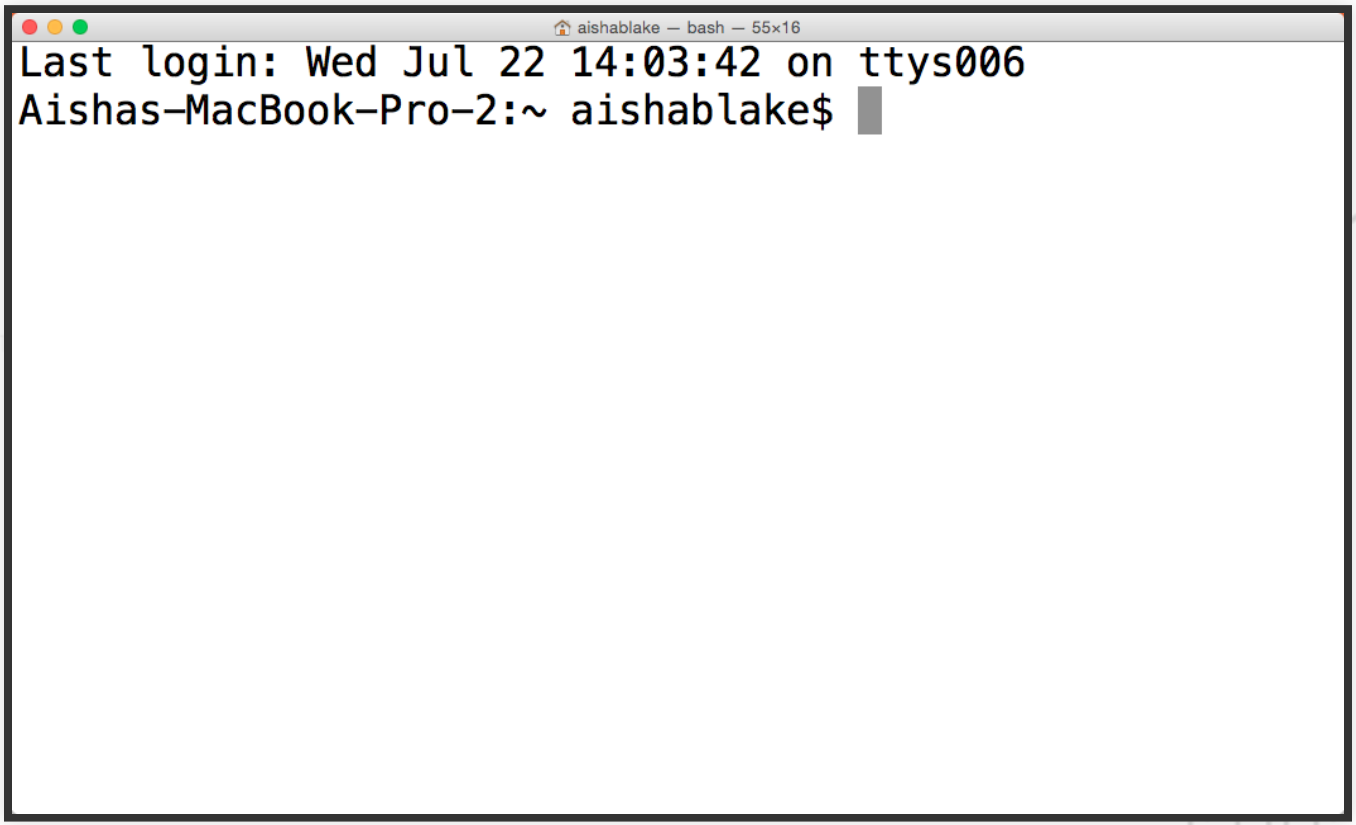
GETTING STARTED

WHICH OS?

Mac users: Open up Terminal

Windows users: Download and install Git Bash

TERMINAL PROMPT



Many command line prompts will end with a dollar sign `$`.

This is your signal that it is okay to type a new command.

TERMINAL CHEAT SHEET

What is a terminal?

What's the point of using a terminal?

Google a cheat sheet if you need one.

NAVIGATION

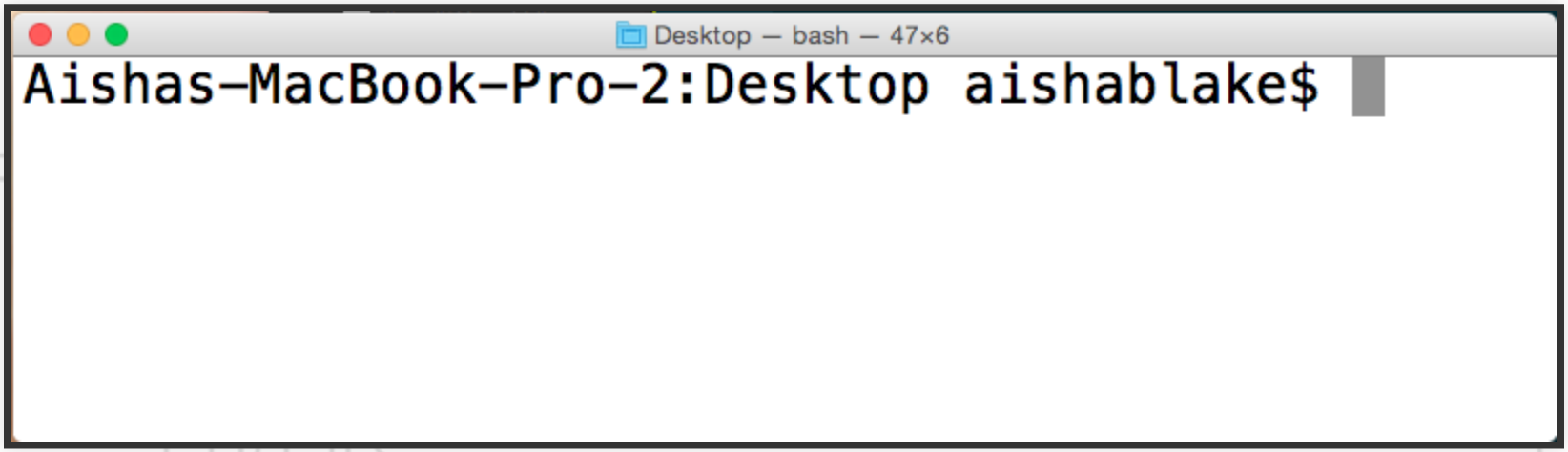
CHANGE DIRECTORY

The `cd` command allows a path to be defined which Terminal will navigate to.

Executing this command will change the current terminal directory to the path you input.

```
cd Desktop
```

GETTING ORIENTED



The Terminal prompt will give some indication as to where it is pointed within the folder structure.

CURRENT DIRECTORY

A single period (.) is used to indicate the current directory.

```
touch ./example.txt
```

PARENT DIRECTORY

Two periods (..) indicates the parent directory.

This is like moving one level "up" within the folder structure.

```
cd ../../another-example.txt
```

ABSOLUTE VS. RELATIVE

ABSOLUTE DIRECTORY PATH

```
cd /Users/ajay
```

RELATIVE DIRECTORY PATH

```
cd ./Users/ajay
```

HOME DIRECTORY

A tilde (~) is used to indicate the home directory.

This is essentially saying "go back to the folder that contains all my personal files"

```
cd ~
```

INFORMATION

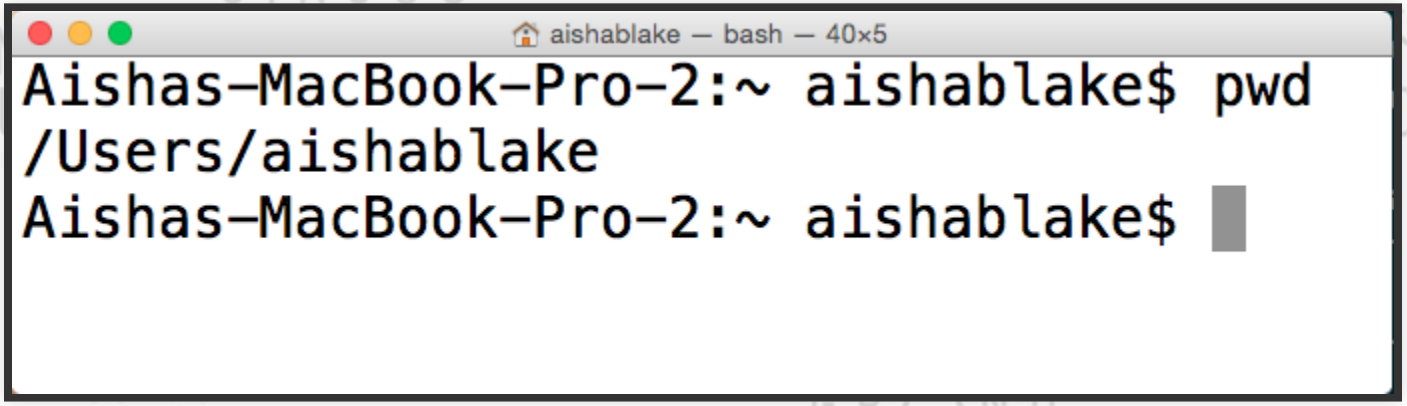
PRESENT WORKING DIRECTORY

The `pwd` command returns a path showing the current directory path of your terminal.

```
pwd
```

On a Windows machine? Try using `cd`.

YOU ARE HERE


aishablake — bash — 40×5
Aishas-MacBook-Pro-2:~ aishablake$ pwd
/Users/aishablake
Aishas-MacBook-Pro-2:~ aishablake$

LIST

Use `ls` to see the contents of the current directory.

```
ls
```

On a Windows machine? Try using `dir`.

CREATION

NEW FILE

Create a new file by typing the `touch` command followed by the name of the file needing to be created.

That file will be added to the current directory.

```
touch new-file.txt
```

NEW FOLDER

Create a new directory (or folder) by typing the `mkdir` command followed by the name of the folder.

That folder will be added to the current directory.

```
mkdir my-stuff
```

PRO TIPS

PREVIOUS COMMANDS

Cycling through any previously executed commands can be done using the up and down arrows.

This is useful for many reasons:

- Correcting small errors in long or complicated commands
- Retyping frequently used commands
- Recalling the steps that led to the current state

MULTIPLE COMMANDS

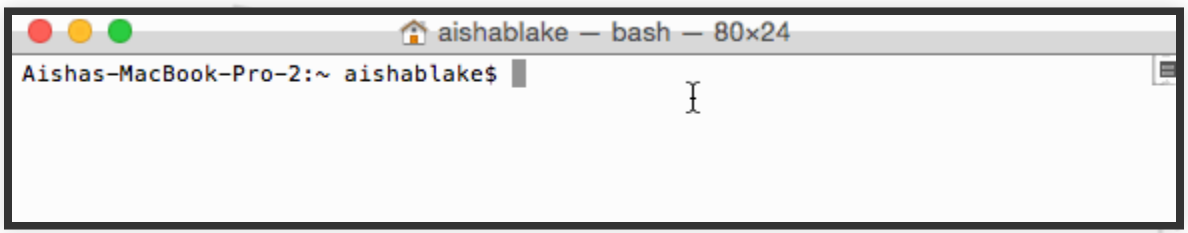
Stringing multiple commands together with two ampersands (&&).

These commands will be executed in order.

```
mkdir my-stuff && cd my-stuff
```

AUTOCOMPLETE

Pressing `tab` will trigger autocomplete.



CLEARING THE SCREEN

The `clear` command (you guessed it) *clears* the terminal window.

We can also use the keyboard shortcut `Cmd+K`.

On a Windows machine? Try using `cls`.

VERSION CONTROL

VERSION CONTROL

Version control (sometimes called source control) is a system that manages changes to a program, website, or other collection of files.

VERSION CONTROL

Version control facilitates two key processes in development:

- Collaboration
- Managing changes to the codebase

COLLABORATION

Version control allows groups to work on the same project simultaneously and helps to avoid and manage the adverse changes.

MANAGING CHANGES

Version control systems are *powerful* because you have full control the lifecycle of your codebase. From *reverting* code that could cause bugs to *adding* code when building new features.

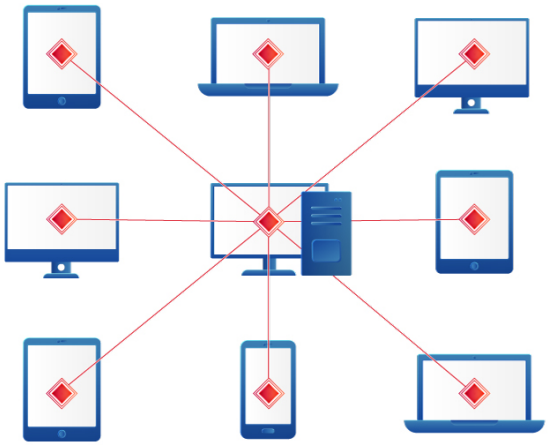
TYPES OF VERSION CONTROL

Kinds of version control:

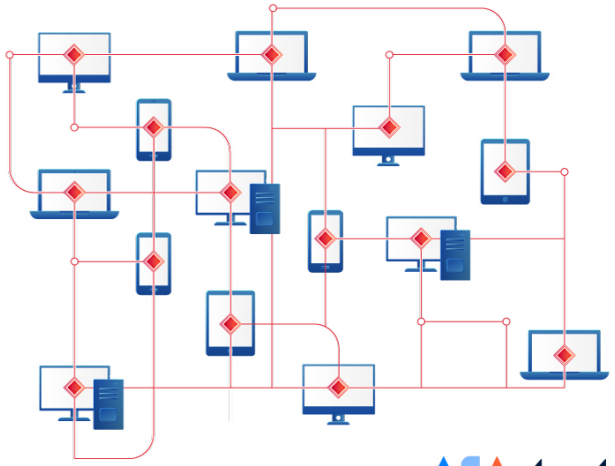
- Centralized
- Distributed

Centralized vs. P2P

Centralized network



P2P network



Lisk ACADEMY

CENTRALIZED

Centralized version control systems are run on one central server infrastructure.

Each collaborator checks out the code from and merges changes into the main server.

DISTRIBUTED

Distributed version control systems allow each collaborator to maintain a separate repository of the code which can be periodically reconciled with other peer repos.

GIT, GITHUB & THE COMMAND LINE


git
≠
github
SOCIAL CODING

GIT & GITHUB

Git is an open source, distributed version control system initially developed by Linus Torvalds.

GitHub is a social coding service that offers hosting for software projects that use Git as their source control.

GIT & GITHUB

The combination of the two has become the gold standard in the OSS community and startup scene.

It is gaining major ground in the enterprise space as well.

GIT

Open your command prompt:

- Git Bash in Windows
- Terminal in OSX

INSTALLATION

VERIFY YOUR INSTALL

```
git --version
```

```
git version 2.4.1
```

IF YOU ARE NOT GOOD

If Git isn't on the machine, download and install it.

Get Git!

Create a GitHub account!

BASIC CONFIGURATION

```
git config --global user.name "Your Name"
# sets the default name for git to use when you make a commit

git config --global user.email "you@email.com"
# sets the default email for git to use when you make a commit
```

GIT COMMANDS

8 GIT COMMANDS YOU NEED

These commands cover most of what is needed to do on a daily basis.

- `git init`
- `git status`
- `git add`
- `git log`
- `git commit -m`
- `git push`
- `git pull`
- `git reset`, `git revert`
- `git clone`

GIT INIT

Creates a new local git repository. A repository is where all your source code lives.

```
cd Desktop    # or wherever you want the code to live
mkdir git-demo && cd git-demo
git init
```

NEW FILES

Create a new file in the new empty repo.

Call it < yourname >.txt

```
touch ajay.txt
```

GIT STATUS

Reports the current status of the repo, such as whether any files have been modified or new files have been created.

```
git status
```

GIT ADD

Adds untracked files to the repo AND adds a tracked file's changes to the staging area, making it ready to be committed.

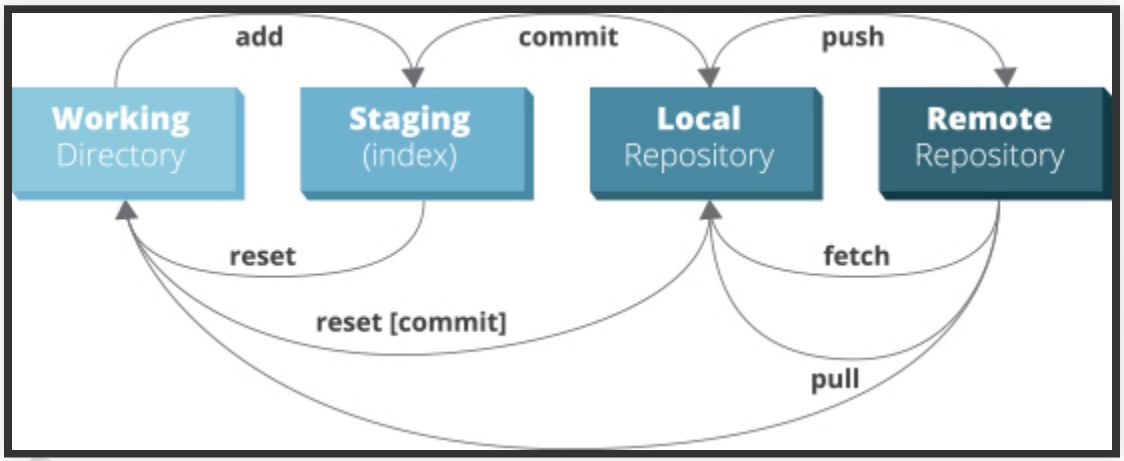
`git add` must be called with a parameter that is the path to the file(s) being added.

```
git add ajay.txt
git status
```

STAGING AREA

There are 4 states a file can exist in a repo.

- Untracked
- Tracked
- Changed
- Staged
- Committed (which is really just back to 'Tracked')


add
commit
push
Working Directory
Staging (index)
Local Repository
Remote Repository
reset
fetch
reset [commit]
pull

GIT ADD

After you add the file, recheck the status.

The file is now being tracked and is ready to be committed.

Make a change to the file and check the status again. What happened?

`git add` the file again.

Check your `git status` again. What happened?

.GITIGNORE

WHY WE IGNORE

Sometimes, there will be private files that need to be kept private.

Other times, files will get generated automatically that do not necessarily need to be kept track of.

THE SOLUTION

We can use a particular file called `.gitignore` to list out everything we don't want Git to track.

You can also use wildcards and directories in `.gitignore`

For example:

```
.DS_Store
src/tests/reports/
*/**/screenshots
```

(more on that later)

GITIGNORE FOR JAVASCRIPT

When we get into building JavaScript applications, an important file to add to .gitignore is `node_modules`.

`node_modules` are third-party dependencies that we want to pull in locally (more on that later).

GIT COMMIT -AM < MESSAGE >

Once a file has been staged, it can be commited.

GIT COMMIT

Commit the new file to the repo.

```
git status
git commit -am "I added a new file!"
```

GIT LOG

`git log` displays the repo commit history.

```
git log
```

Pressing `q` will exit the log.

There are tons of options you can add to the `git log` command to customize how it looks.

GIT RESET

`git reset` is used to undo commits but it can rewrite commit history, so be careful.

`git reset` must be used with caution.

Add text to `git-test.txt`, save it, add it, commit it.

```
git reset --hard SHA
git status
```

This will also delete all the commits and reset to the commit of your choosing

GIT REVERT

`git revert` is used to undo a commit by creating a new commit.

This is the preferred way of undoing bad things, as it does not re-write the repo's commit history.

Add text to `git-test.txt`, save it, add it, commit it, check the history with `git log`.

```
git log
git revert --no-commit SHA..HEAD
git status
```

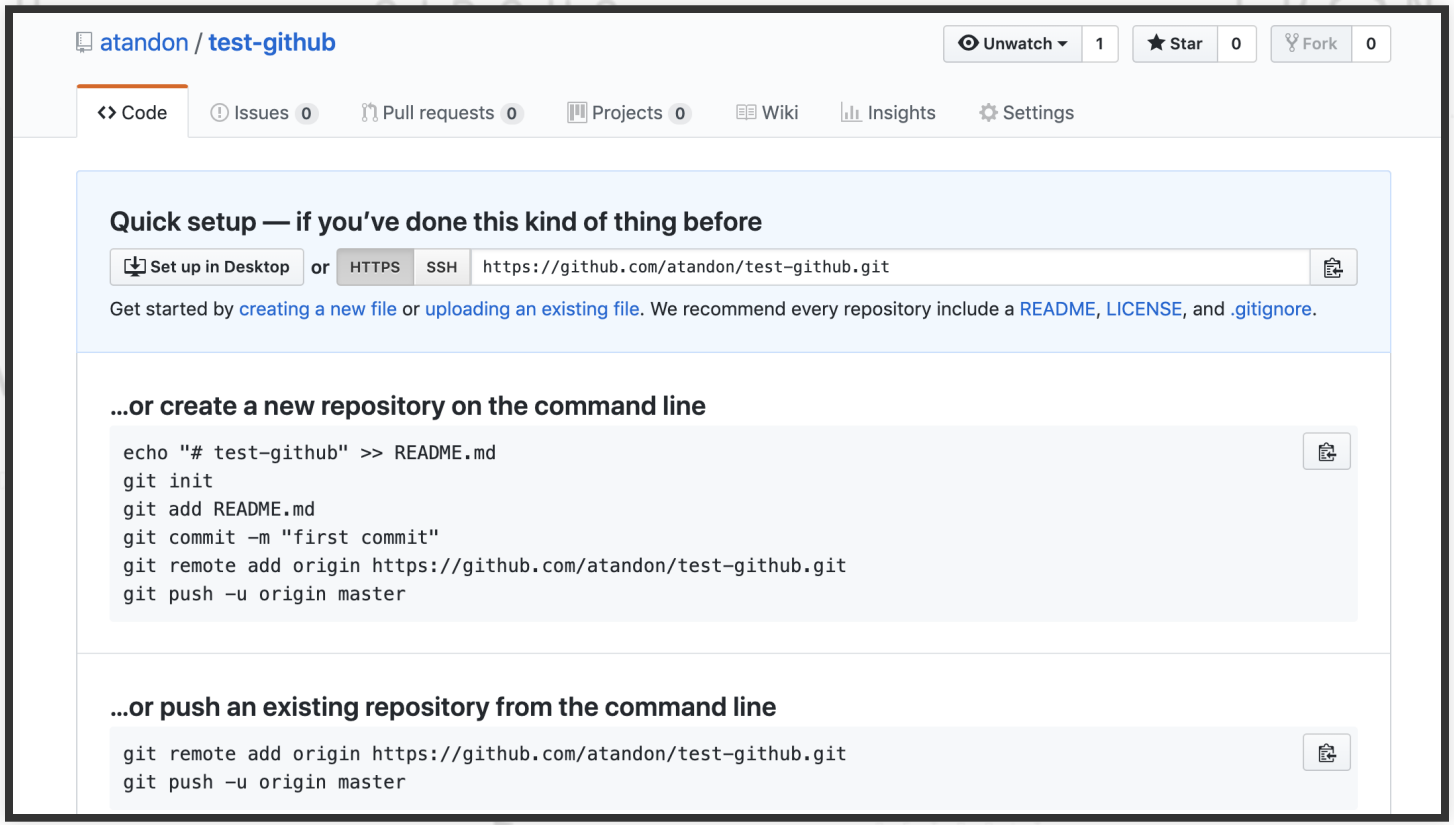
GITHUB

GITHUB

Where all your code lives remotely!

- Easy Collaboration (Code reviewing, OSS, Analytics)
- Backs up all your code
- So much more...

CONNECTING GITHUB TO YOUR NEW LOCAL REPOSITORY



LET YOUR REPO KNOW ABOUT YOUR NEW REMOTE REPOSITORY

```
git remote add origin
https://github.com/atandon/test-github.git
```

PUSHING TO A REMOTE

Once any changes are committed locally and a remote repository has been created, `git push` can be used to push code to that repo.

PUSHING TO A REMOTE

`git push` pushes a local repo to a remote location.

```
git push origin master
# or
git push
```

PULLING FROM A REMOTE

`git pull` pulls changes from a remote repo and attempts to merge them with the local changes.

The local repo must be in sync with the latest changes to a remote before new changes can be pushed to it.

```
git pull origin master
# or if you want to pull changes from all branches
git pull
```

CLONING A REPO

`git clone` creates a local copy of a remote repository in your local file system allowing you to make local changes.

Note: This will make a new directory of the repo's root folder whenever you run this command.

```
git clone https://github.com/atandon/test-repo
```

BRANCHING

BRANCHING

- A repository usually has a default branch, called `master`, a stable version of your application + ready for the public to use.
- For the development of new features, such as implementing a new login form, a new branch should be created off of master.

GIT BRANCH

A branch can be created on the repository's page on GitHub, but can also be created in your Terminal.

The branch must be named during creation.

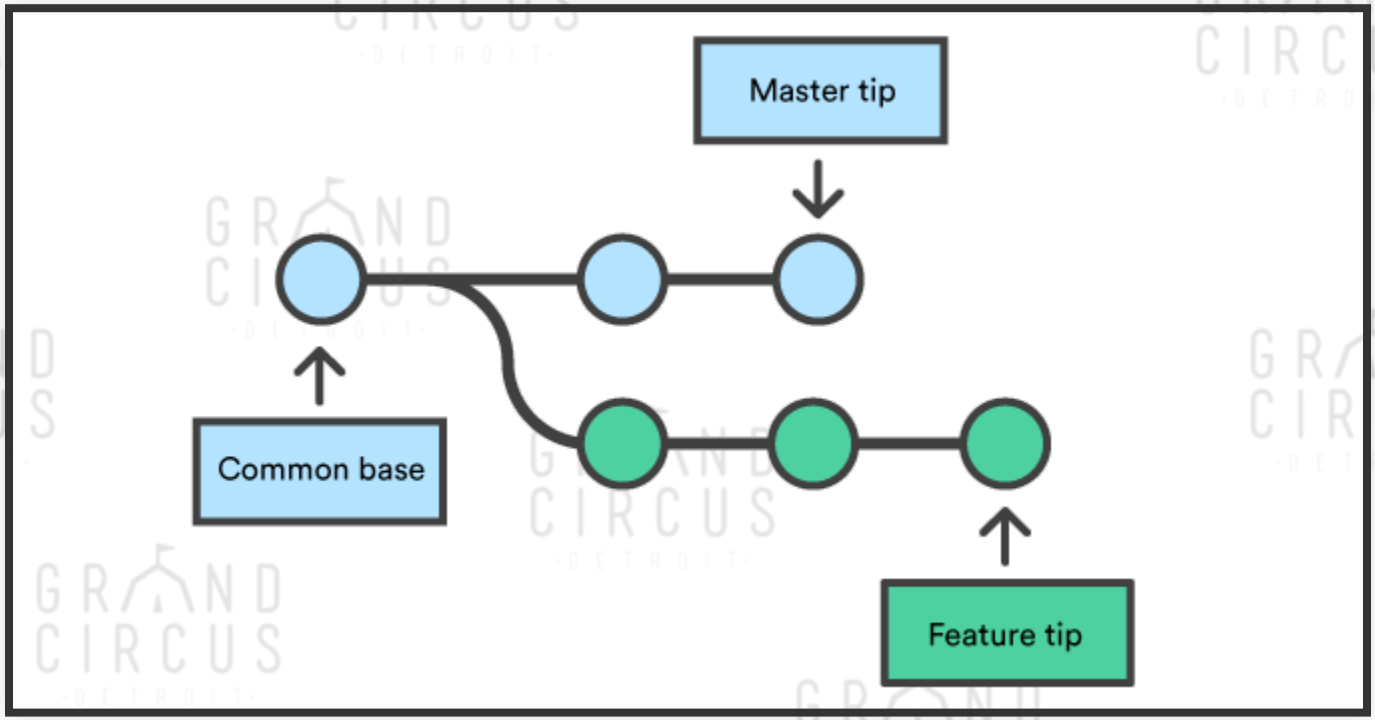
The name of the branch should reflect what the branch is trying to accomplish.

```
git branch login-form
```

GIT BRANCH

It will take the *current* state of `master` and duplicate it under the branch named `login-form`.

This new branch is a clone of the `master`.


Master tip
Common base
Feature tip

GIT CHECKOUT

After a branch has been created, Git needs to be switched over to that branch.

The technical term for changing branches is known as `checkout`.

To checkout to a different branch, use the following command:

```
git checkout branch-name
```

GIT BRANCH & CHECKOUT

Because terminal allows multiple commands to run on a single line, creating and swapping to a new branch is fairly simple.

```
git checkout -b login-form
```

MERGING

MERGING

You finished creating the login form and are ready to merge. What happens?

To merge branches, `git checkout` to the branch the new features should be merged to.

In this case, you want to merge `login-form` into the `master` branch.

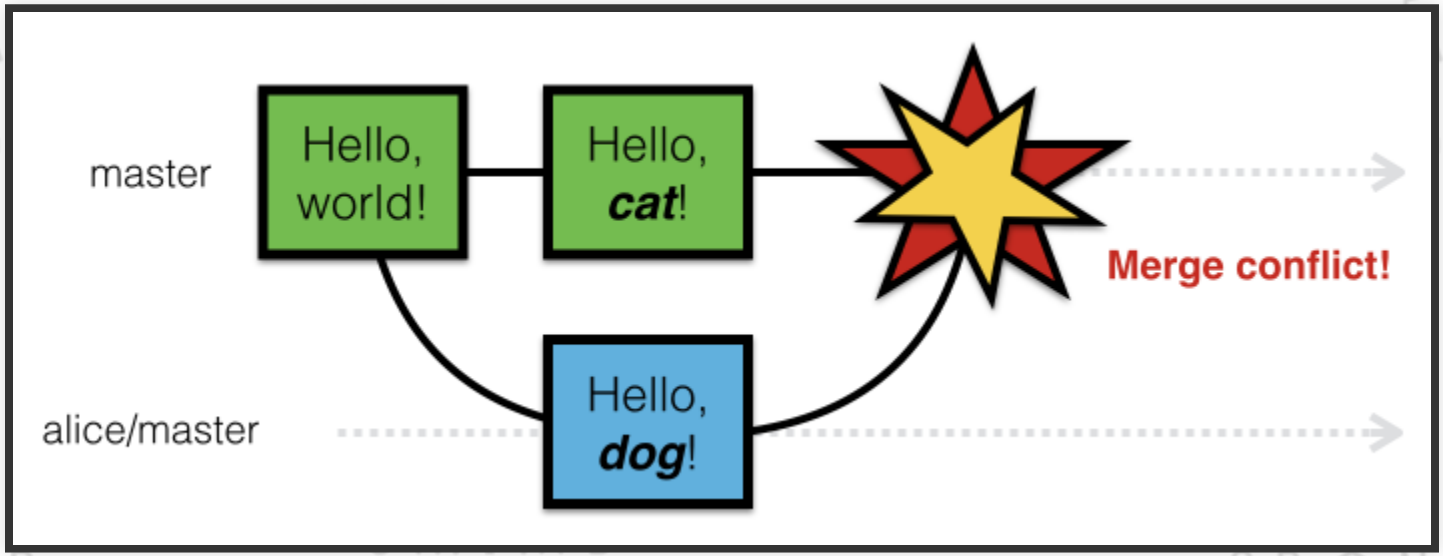
```
git checkout master
git merge login-form
```

MERGE CONFLICTS

Merge conflicts arise when a developer attempts to merge code to a branch with competing changes to the same lines of code.

Git won't know which line edit should be used.

Do not panic, these conflicts are relatively easy to handle.


master
Hello, world!
Hello, *cat*!
Merge conflict!
alice/master
Hello, *dog*!

MERGE CONFLICTS

Whenever there is a merge conflict, Git will notify of the problem through Terminal.

Git will display what files have merge conflicts and that they will need to fix them.

For a simple merge conflict, it may look like this:

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

MERGE CONFLICTS

Fixing a merge conflict is going to require the file(s) to be altered.

At minimum, two pieces of code will be presented...something called `HEAD` and then the modified changes.

```
<<<<<<< HEAD
function login() { console.log("this is temporary"); }
=======
function login() {
  let userInfo = {
    name: document.querySelectorAll(".username")[0].value,
    password: document.querySelectorAll(".password")[0].value
  };
  LoginService.checkCredentials(userInfo);
}
>>>>>>> 3ddebc34e7ede82b59d421d01122cdff9a0f6b66
```

MERGE CONFLICTS

Once the decision has been made of which pieces of code should be kept, simply just delete what you do not want to keep.

For example, if the login function is what we want, we can delete the following code:

```
<<<<<<< HEAD
function login() { console.log("this is temporary"); }
=======

>>>>>>> 3ddebc34e7ede82b59d421d01122cdff9a0f6b66
```

MERGE CONFLICTS

The only thing left in script.js now is:

```
function login() {
  let userInfo = {
    name: document.querySelectorAll(".username")[0].value,
    password: document.querySelectorAll(".password")[0].value
  };
  LoginService.checkCredentials(userInfo);
}
```

FORKING

FORKING

Developers will inevitably be working on a team at some point.

`Forking` allows people to create a copy of a repository, make any changes they want to the local version, commit, push, and then submit what is called a `pull request`.

The process of forking is quite straightforward.

There are no tricks or hidden magic, follow the steps and you are good to go.

FORKING

Start off by heading over to a repository.

There is a button which says "FORK" in the upper right corner, click it.

A window will display asking where to fork the repository.

Click your username. When it is done, you will now have your own copy of the repository.

FORKING

Go through the normal process of cloning this repository.

Create a branch in your local repository, make some changes, add/commit/push.

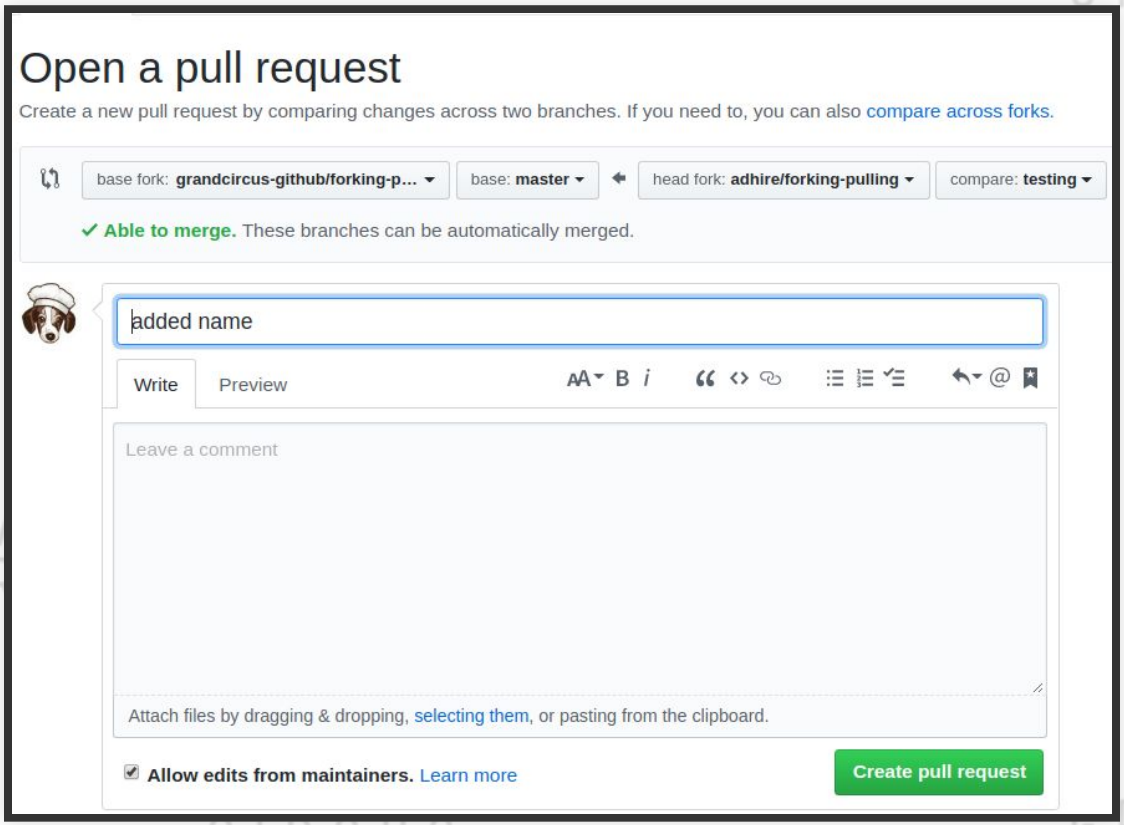
FORKING

Refresh the browser page, select the branch being worked on, click "New Pull Request".

This will take direct you to a new page that looks similar to the next slide.

PULL REQUESTS

PULL REQUESTS



PULL REQUESTS

The left two inputs represent the repository and branch content is being merged INTO.

The right two inputs represent the repository and branch content is being merged FROM.

Leave a comment to describe what is being altered.

Click on "Create a pull request".

PULL REQUESTS

The owner of the original repository is notified about the pull request and can check out what the pull request changes.

They can accept it or reject it.

This is how collaborating with others works using Git/GitHub.

For group projects, spend time using pull requests. This is going to save much time in the long run.

STAYING UPDATED

Type the following command:

```
git remote -v
```

This is displaying where content will be pulled from and pushed to.

When forking, it is smart to establish an upstream that will be synced to the forked repo.

```
git remote add upstream git://github.com/ORIGINAL_USERNAME/REPO_NAME.git
```

STAYING UPDATED

If the repository you have forked receives updates, you will need your local repository to receive those updates.

```
git fetch upstream
git checkout master
git merge upstream/master
```

Now your local master branch has the updated repo.

Carry on with your normal process.