GIT & GITHUB

INTRO TO THE COMMAND LINE

GETTING STARTED

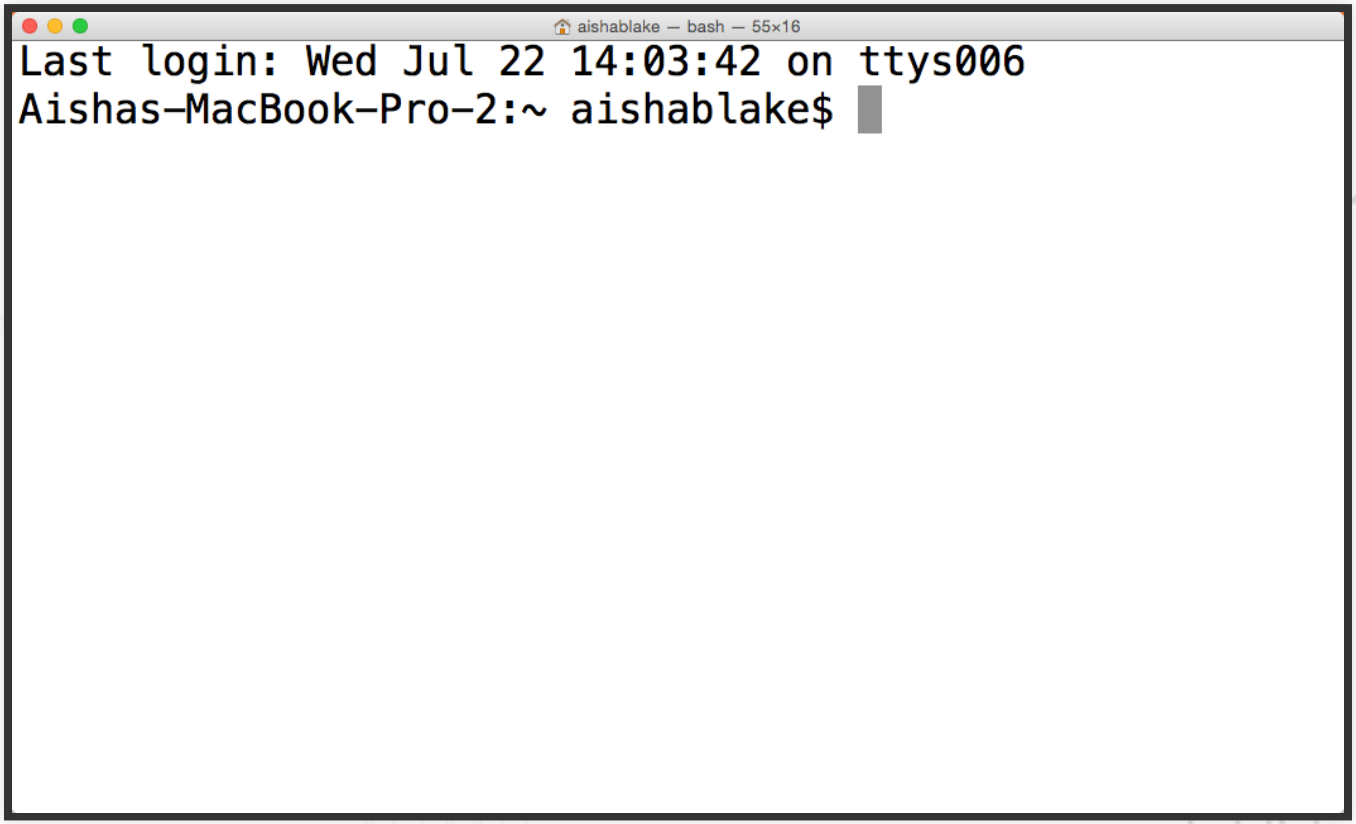
WHICH OS?

Mac users: Open up Terminal

Windows users: Download and install Git Bash

If you're on a PC, you can use the command line tool that came with your computer, but know that certain commands will be different.

TERMINAL PROMPT



Many command line prompts will end with a dollar sign `$`. This is your signal that it's okay to type a new command.

TERMINAL CHEAT SHEET

There are lots of things we can do with the command line. Once you get used to it, doing things this way can be a lot faster than switching between the mouse and the keyboard, pointing and clicking.

We're going to go over a few very important commands that will be particularly useful to us, but you should feel free to refer back to this cheat sheet as often as necessary.

Windows users can use this cheat sheet.

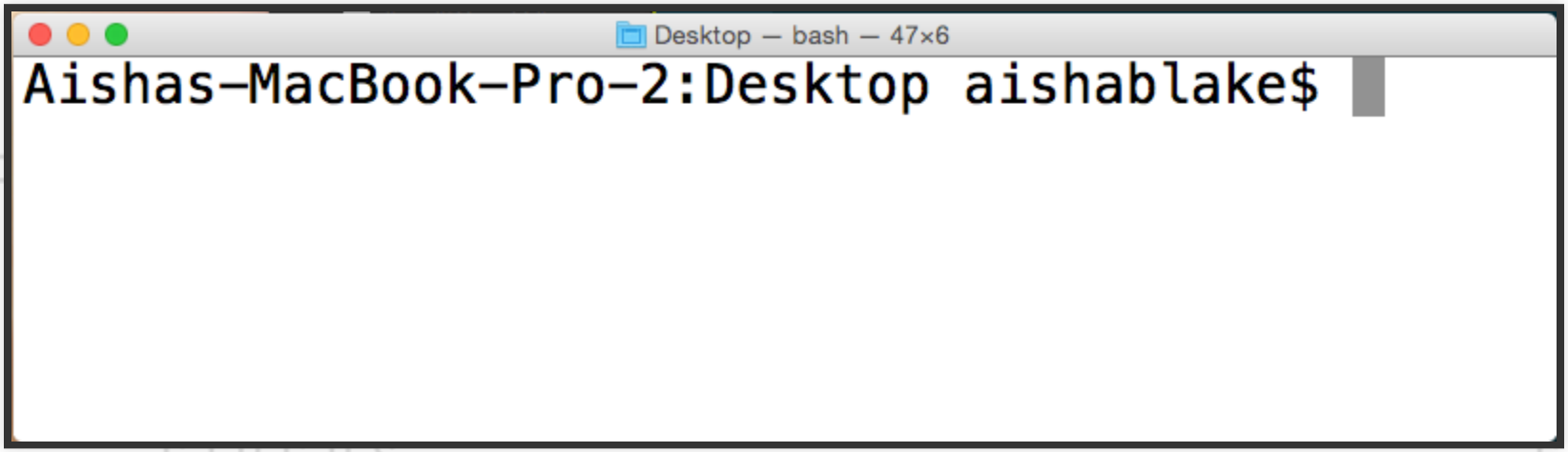
NAVIGATION

CHANGE DIRECTORY

The `cd` command allows us to define a path to the directory we wish to navigate to. Executing this command will change where we are in the folder structure. This change should be reflected in our command prompt.

```
cd Desktop
```

GETTING ORIENTED



The terminal prompt will give us some indication as to where we are 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
```

ROOT DIRECTORY

A tilde (~) is used to indicate the root directory. We're essentially saying "go back to the beginning and start from scratch". On a Mac, this is your Home.

```
cd ~
```

INFORMATION

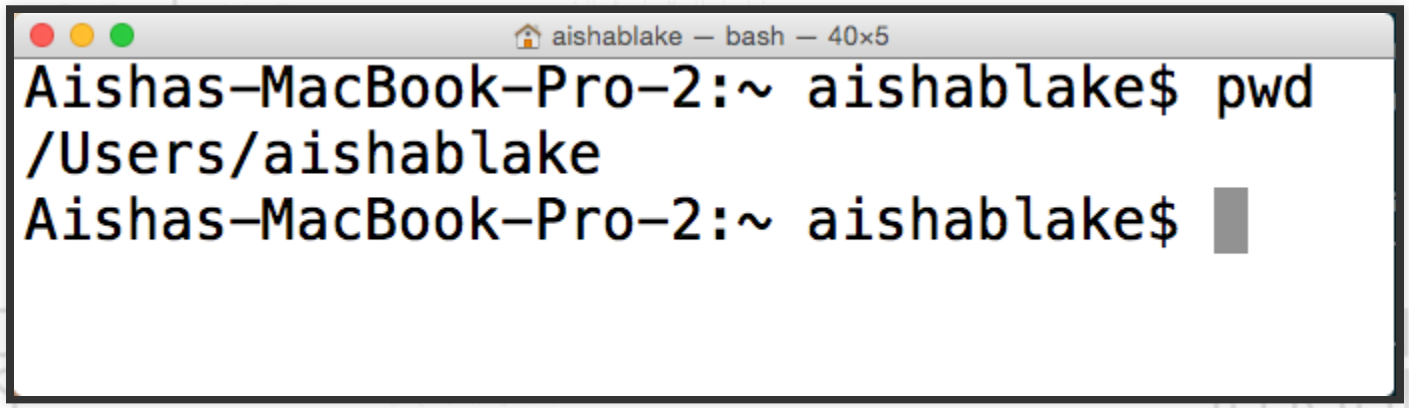
PRESENT WORKING DIRECTORY

The `pwd` command returns a path showing exactly where we are within the folder structure.

```
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$

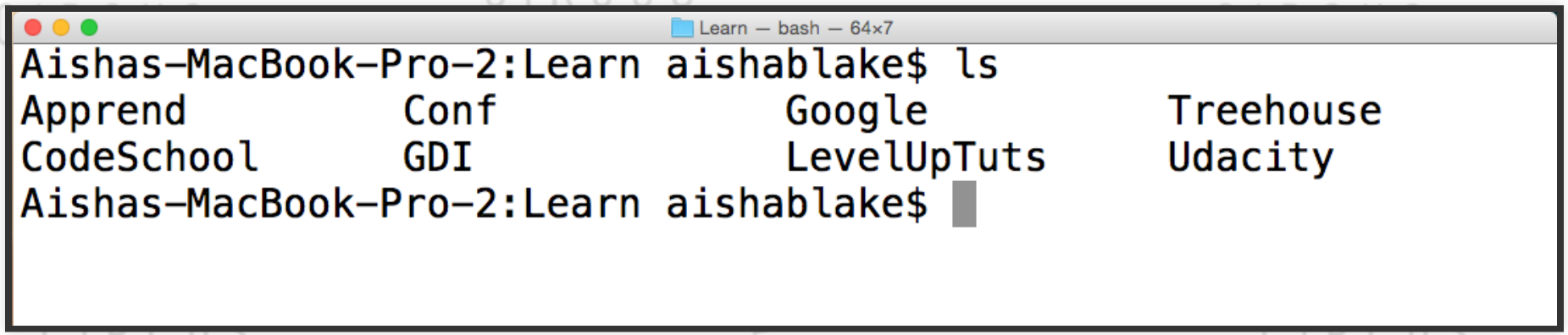
LIST

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

```
ls
```

On a Windows machine? Try using `dir`.

TABLE OF CONTENTS



HELP

When you're not sure what your options are, typing `help` will list all possible commands.

```
help
```

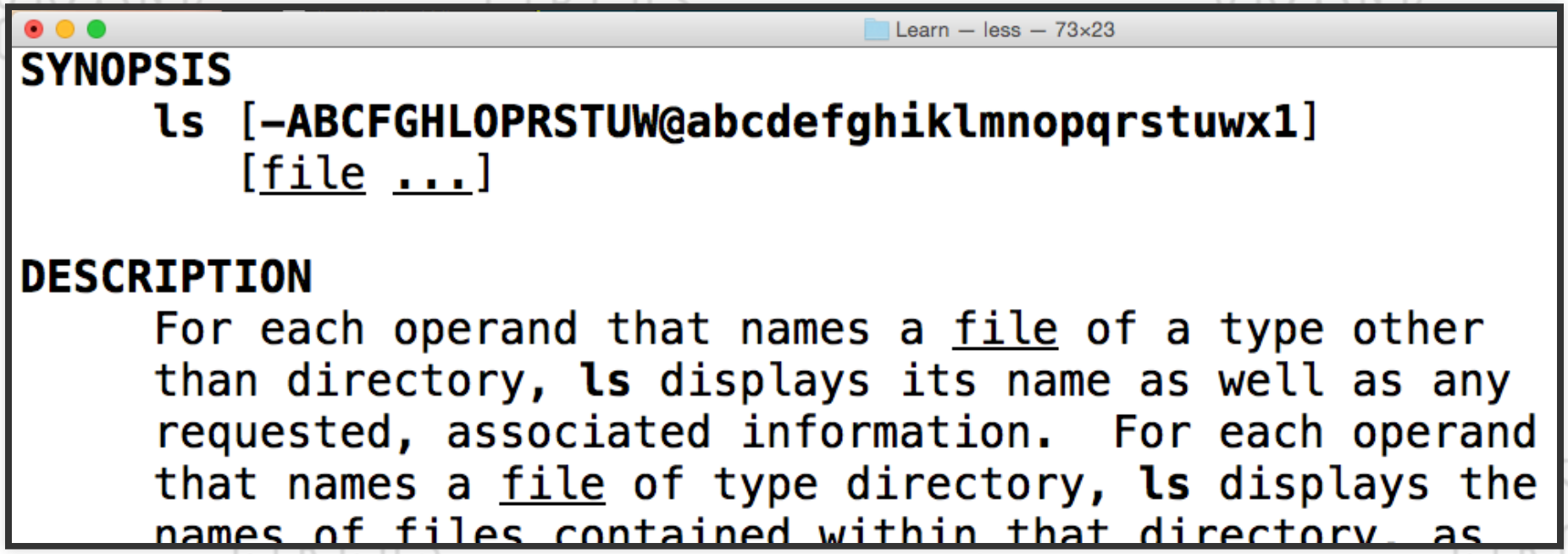
MANUAL

You've figured out what you *can* say, but not what the commands do. Use `man` with any command to see the manual for that command. Hit the 'q' key to escape.

```
man ls
```

On a Windows machine? Try using `help`.

NOBODY READS THE MANUAL



CREATION

NEW FILE

Create a new file by typing the `touch` command followed by the name of the file you wish to create. That file will be added to the current dirctory.

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

On a Windows machine? Trying using `echo [some text] > new-file.txt`.

NEW FOLDER

Create a new directory (or folder) by typing the `mkdir` command followed by the name of the folder you wish to create. That folder will be added to the current directory.

```
mkdir my-stuff
```

PRO TIPS

PREVIOUS COMMANDS

We can cycle through any previously executed commands using the up and down arrows. This is useful for a number of reasons:

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

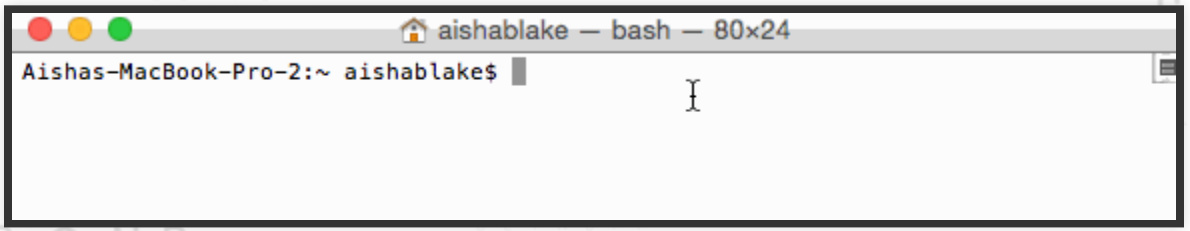
MULTIPLE COMMANDS

We can string multiple commands together with two ampersands (&&). These commands will be executed in order.

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

AUTOCOMPLETE

Pressing `tab` once you've started typing a directory or file name 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 you to work on the same project simultaneously with other people and helps to avoid and manage the conflicting changes.

MANAGING CHANGES

Version control systems allow for seamless integration of new code, easy viewing of previous changes, and makes it easy to undo a change set. It also allows us to revert documents back to a previous state.

TYPES OF VERSION CONTROL

Kinds of version control:

- Centralized
- Distributed

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.

VOCABULARY

WHAT DOES IT ALL MEAN?!

We've perhaps already thrown around a couple of words that didn't make sense at the time. Well, now is the time for all to become clear!

Note Remember that you can (and should) always just ask if we say something you don't understand!

REPO

A location where data is stored and managed

UNTRACKED

When we talk about "untracked" files, we mean that Git isn't tracking the changes made to such files. It's aware that they exist, but that's about it.

TRACKED

Tracked files are monitored by Git. It keeps tabs on every little change made to tracked files.

GIT, GITHUB & THE COMMAND LINE


git
≠
github
SOCIAL CODING

GIT & GITHUB

Git is an open source, distributed version control system originally 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 have 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
```

If you don't get some kind of error, you're good. git-version

IF YOU'RE NOT GOOD

If you've never used Git before, you may need to download it. If you're on a Mac, you'll already have Git installed out of the box but may want to upgrade to the latest stable version.

Get Git!

While we're at it, if you haven't already...

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
```

.GITIGNORE

WHY WE IGNORE

Sometimes, there will be private files that you want to keep private. Other times, you'll have files that get generated automatically that you don't necessarily want to have to keep track of.

THE SOLUTION

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

A good item to toss in there right now is `.DS_Store`.

.GITIGNORE FOR JAVA

For right now, the only Java-specific thing we'll add to the `.gitignore` file is your `.class` files. Now your `.gitignore` should look something like this:

```
.DS_Store
*.class
```

GIT COMMANDS

8 GIT COMMANDS YOU NEED

These cover most of what you need to do on a daily basis.

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

GIT INIT

Creates a new git repository in the current folder including all child folders.

```
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 aisha.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 paramater that is the path to the file(s) you wish to add.

```
git add aisha.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')

GIT ADD

- After you add the file, check your status again. You should see that the file is now being tracked and is ready to be committed.
- Change the file in your editor in some way and save it. Run another `git status`. What happened?
- `git add` the file again. Check your `git status` again. What happened?

GIT COMMIT -M < MESSAGE >

Once you have code staged, you can commit the change to your repo. This will create an anchor point that you can build from or return to if needed.

Protip: Don't forget the `-m`. If you do, you will be thrown into an in-terminal editor and that will ruin your whole day.

GIT COMMIT

Commit the new file to your git repo.

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

I check my status like a crazy person. Usually to make sure I'm not committing something I don't want to. `git status` will be one of your most used commands.

VIM

If you do find yourself in the command line editor, type this sequence of four keys to get out.

- *Escape* exits edit mode, so you're not just typing in the file.
- *: (colon)* starts a command. You'll see your cursor at the bottom of the screen.
- *x* is the command to save and quit the editor.
- *Enter* runs the command.

GIT LOG

`git log` lets you view your repo's commit history.

```
git log --pretty=format:"%h - %an, %ar : %s"
```

(you may need to press 'q' to exit the log) There are tons of options you can add to the `git log` command to customize how it looks.

GITHUB

GITHUB

In order to push our code to GitHub, we will need to create a remote repository for it to live.

Code with me!

PUSHING TO A REMOTE

Once your changes are commited locally and you have a remote repository you can push to, you can do a `git push` to push your 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 your local changes. You must be in sync with the latest changes to a remote before you can push changes up to it.

```
git pull origin master
# or
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.

(do this in another directory, not your current project)

```
git clone https://github.com/kroysemaj/git-demo
```

UNDOING BAD THINGS

UNDOING BAD THINGS

It is inevitable that at some point you will make a mistake coding. Whether it is breaking a function, accidentally deleting something, or simply just needing to go back to a previous commit.

Git has a great toolset to help us out. We will take a look at three options: `git checkout`, `git revert`, and `git reset`.

Create a file named "git-test.txt" for some practice.

GIT CHECKOUT

`git checkout` is used if you have altered something prior to `git add`. You can use `git checkout` to restore a file to what it was before you altered it.

Add your name to `git-test.txt` and save it. Run the following command and then see if `git-test.txt` contains the data you just saved.

```
git checkout git-test.txt
```

GIT REVERT

`git revert` is used undo a commit by creating a new commit. This is the preferred way to undoing bad things, as it doesn't re-write your history.

Add text to `git-test.txt`, save it, add it, commit it. We will use `git log` to find a commit in the log to revert to.

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

Now we just need to make a commit and we're done.

GIT RESET

`git reset` is used to undo commits but it re-writes your commit history. This should never be used with a public repository...and even your own private repository should be treated with caution.

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

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

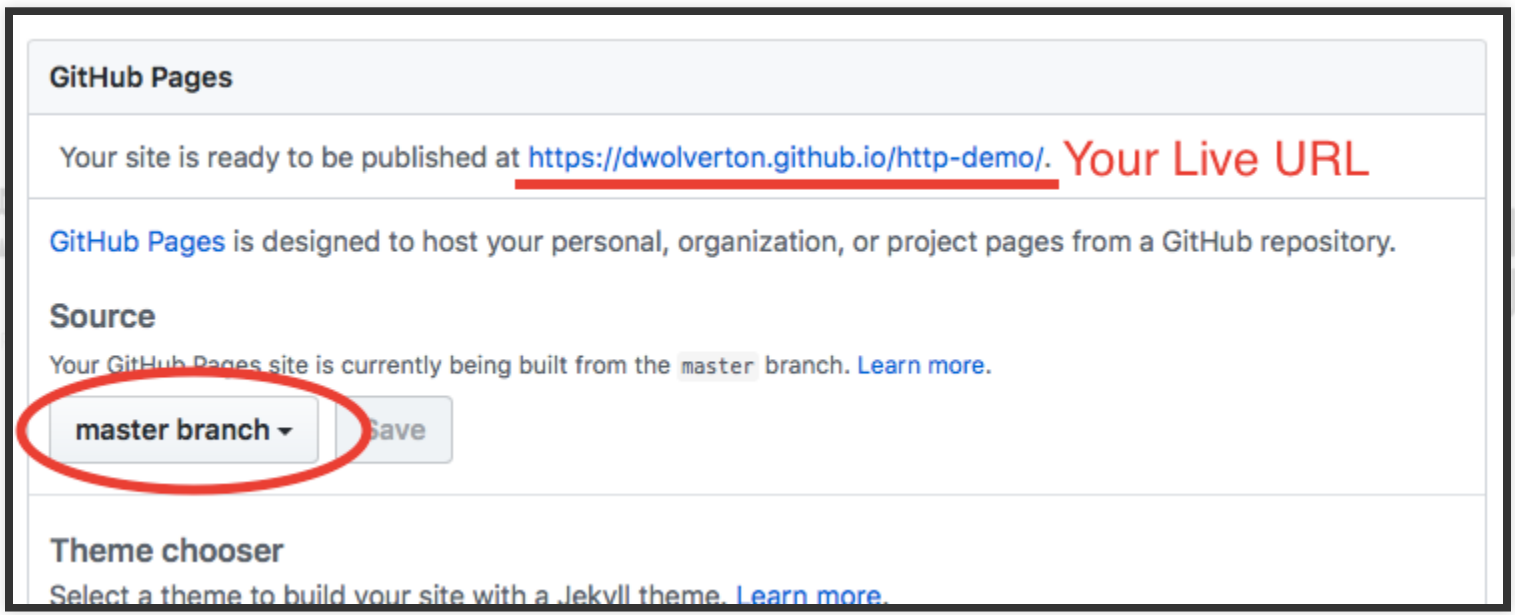
This will also delete all the commits and reset to the commit of your choosing

GITHUB PAGES

LIVE HOSTED AND FREE

GITHUB PAGES

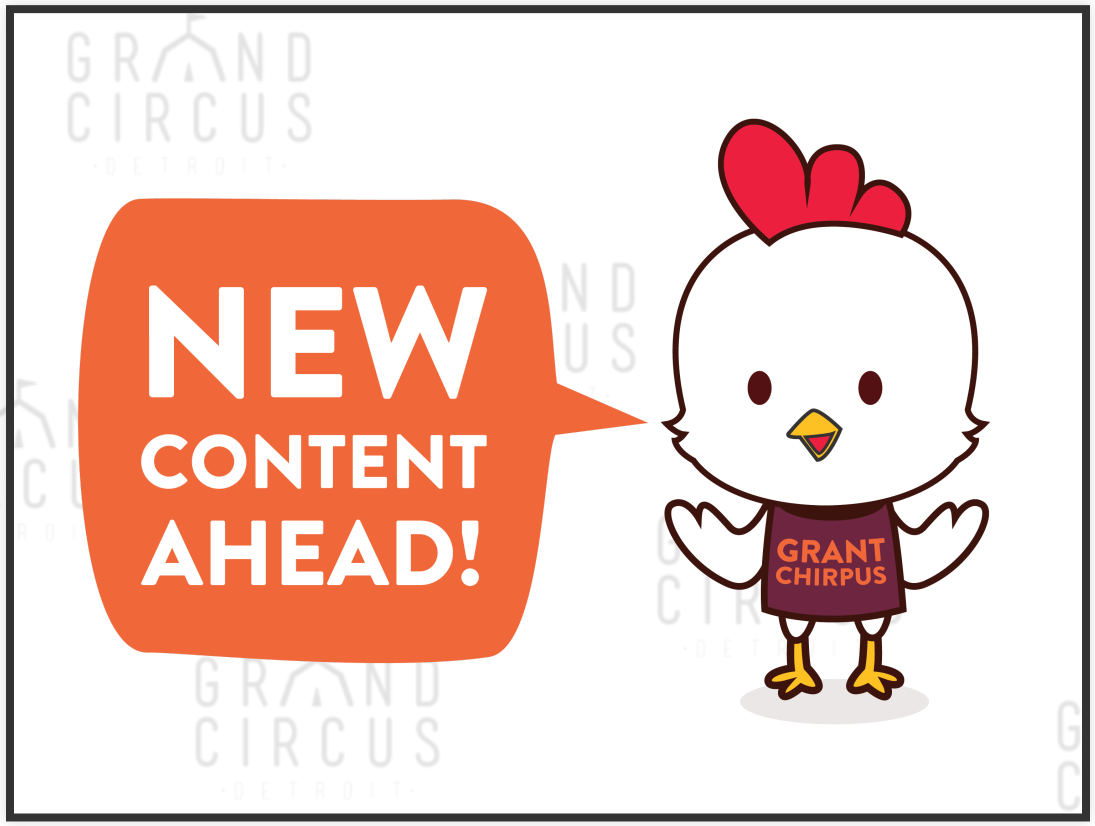
Settings tab at the top of your repo screen on GitHub.com



GITHUB PAGES

This will only work well if you have an HTML file named exactly *index.html*. You should always use this name for your main file.

It also may take up to 10 or so minutes for their servers to host. Do not panic.


NEW
CONTENT
AHEAD!
GRANT
CHIRPUS

BRANCHING

BRANCHING

BRANCHING

A repository is set to have one main branch called `master`. This should be considered the branch that contains the most up-to-date code that contains no bugs or issues.

For development of new features, such as implementing a new login form, a new branch should be created and used. Once the feature is working with no bugs, it can be merged into the `master` for a flawless integration.

GIT BRANCH

You can create branches on the repository's page on GitHub but you can also create it through your terminal. When creating a branch, we need to give it a name. The name of the branch should reflect what the branch is trying to accomplish.

```
git branch login-form
```

GIT BRANCH

If you are on `master` and create a branch called `login-form` it will take the *current* state of `master` and duplicate it under the branch name `login-form`.

Your new branch should be just as up to date as `master`, which will be a big help in developing new features.

GIT CHECKOUT

After a branch has been created, you will need to switch over to that branch. The technical term for changing branches is known as `checkout`.

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

```
git checkout branch-name
```

GIT BRANCH & CHECKOUT

Because terminal allows muliple commands to be ran on a single line, you can create a branch and do the checkout in one line.

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

MERGING

MERGING

At some point, you will finish the new feature and will want to merge that code into your `master` branch.

To merge branches, you will want to `checkout` to the branch you want the new features merged into. Once you are on the branch, you will merge the branch with the new feature in. Let's take a look...

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

MERGE CONFLICTS

During a merge, you may run into something known as a `merge conflict`. If the same file has been altered on the same line(s) on two different branches, a merge conflict will arise.

Git won't know how to handle this so it will defer to your judgement.

MERGE CONFLICTS

Whenever there is a merge conflict, Git will notify you through your terminal. Git will tell you what files have merge conflicts and that they will need to fix them, git add, and make a commit.

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 us to open the file(s) inflicted with conflicts. We will be given two pieces of code...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 you and/or your team decide on which pieces of code you want to keep, you simply just delete what you don't want. For example, if we wanted to keep the login function we created on the `login-form` branch, we can simply delete the following code:

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

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

MERGE CONFLICTS

The only thing left in our 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

You will inevitably be working on a team at some point in the very near future. `Forking` allows you to create a copy of a repository, make any changes you want to the repository locally, commit, push, and then submit what is called a `pull request`.

The process of forking is quite straight forward. There are no tricks or hidden magic, just simply follow the steps and you are good to go.

FORKING

Let's start off by heading over to this repository:

DEMO REPO

Once on the page, in the upper hand corner you will see a button named "FORK". When you click that button, 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

Now we need to go through the normal process of cloning this repository. After the clone, we need to set the upstream of the repository to the original, and grab any updates in case some changes to the orignal master have been made.

```
git remote add upstream https://github.com/original-user-name/repository-name.git
git fetch upstream
```

FORKING

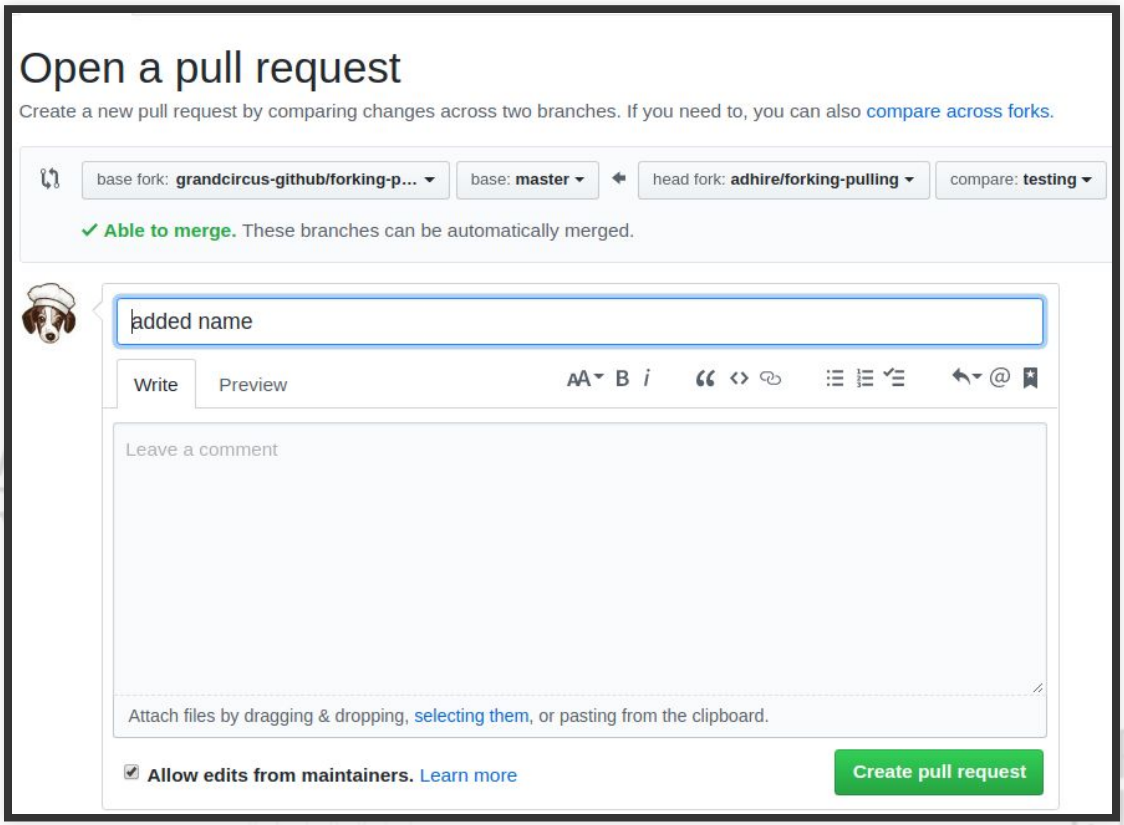
Now, create a branch with your name and alter the student-names.txt file inside. Once you are done, simply just:

```
git add .
git commit -m "added my name"
git push origin branch-name
```

In your browser, refresh the page and select the branch you were working on, and click "New Pull Request". This will take you to a new page that looks similar to the next slide.

PULL REQUESTS

PULL REQUESTS



PULL REQUESTS

The left two inputs are what repository and branch you are trying to merge INTO. The right two inputs are what repository and branch you are trying to merge FROM.

You can leave a comment if you'd like, but ultimately you will want to click "Create a pull request".

PULL REQUESTS

The owner of the original repository is notified about the pull request and can go to GitHub to check out what the pull request is going to do. They can accept it or reject it.

This is how you should be collaborating with others in a group during bootcamp. For group projects, spend time using pull requests. This is going to save a lot of time in the long run.