VERSION CONTROL

WITH GIT

adrian price-whelan

WHAT WE WANT WHEN WRITING CODE

Backups

Ability to revert to a previous version of code

WHAT WE WANT WHEN WRITING CODE

Backups

Ability to revert to a previous version of code

my_script_v1.py

my_script_v2.py

my_script_v3.py

...

not sustainable!

WHAT WE WANT WHEN WRITING CODE

Backups

Ability to revert to a previous version of code

Access your code from anywhere

Share your code

Synchronize changes to code across computers

Mark / release code that works / is stable

VERSION CONTROL TOOLS

Version control systems (software):

Git (git) - widely used, common in astronomy

Mercurial (hg) - still used occasionally

Subversion (svn) - older, probably won't encounter?

VERSION CONTROL TOOLS

Web interfaces / remote storage:

Git - GitHub - https://github.com

Mercurial - BitBucket - https://bitbucket.com

Subversion - Trac - no global repository

GIT CONCEPTS

Create a ***repository*** to store files, directories

Git will *help* keep track of changes to these files, but you have to:

a) tell Git what files to track

b) commit your changes as you make them

Can then ***revert*** changes if necessary, view history of code

GIT CONCEPTS

The full repository can be ***pushed*** to remote sources (e.g., GitHub, external Git server)

> This acts like syncing – your changes (and only your changes) are sent to an external source

If changes are made elsewhere, changes can be ***pulled*** down from the remote source

GIT CONCEPTS

Let's say we start with a folder "`project`" that contains a few files, and we would like to create and add them to a Git repository

```
project/
    my_project/
        file1.py
        tests/
            test1.py
    README.md
```

GIT CONCEPTS

We first change to the `project` directory and initialize an empty repository

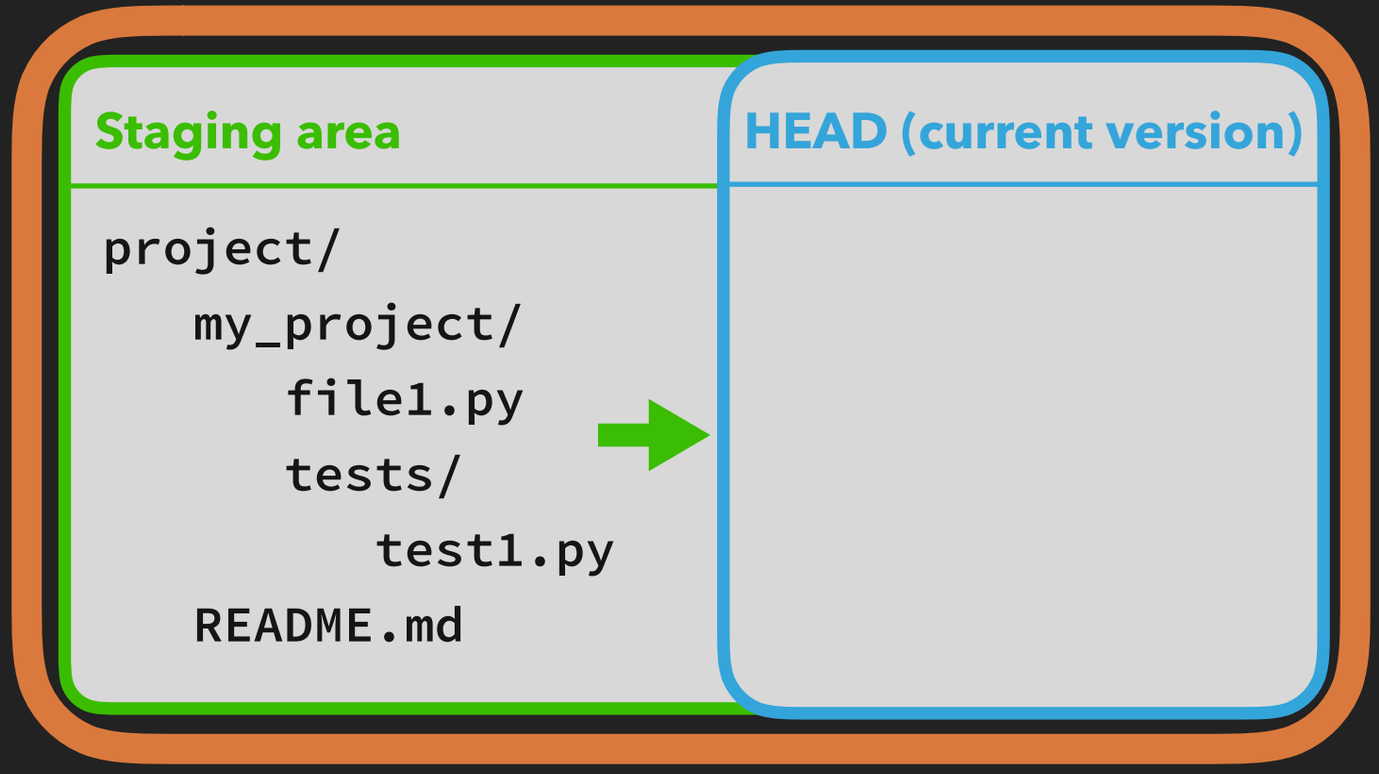
```
> cd project
> git init
```

```
project/
    my_project/
        file1.py
        tests/
            test1.py
    README.md
```

GIT CONCEPTS

We then have to explicitly add and commit these files to the repository

```
> git add *
```



GIT CONCEPTS

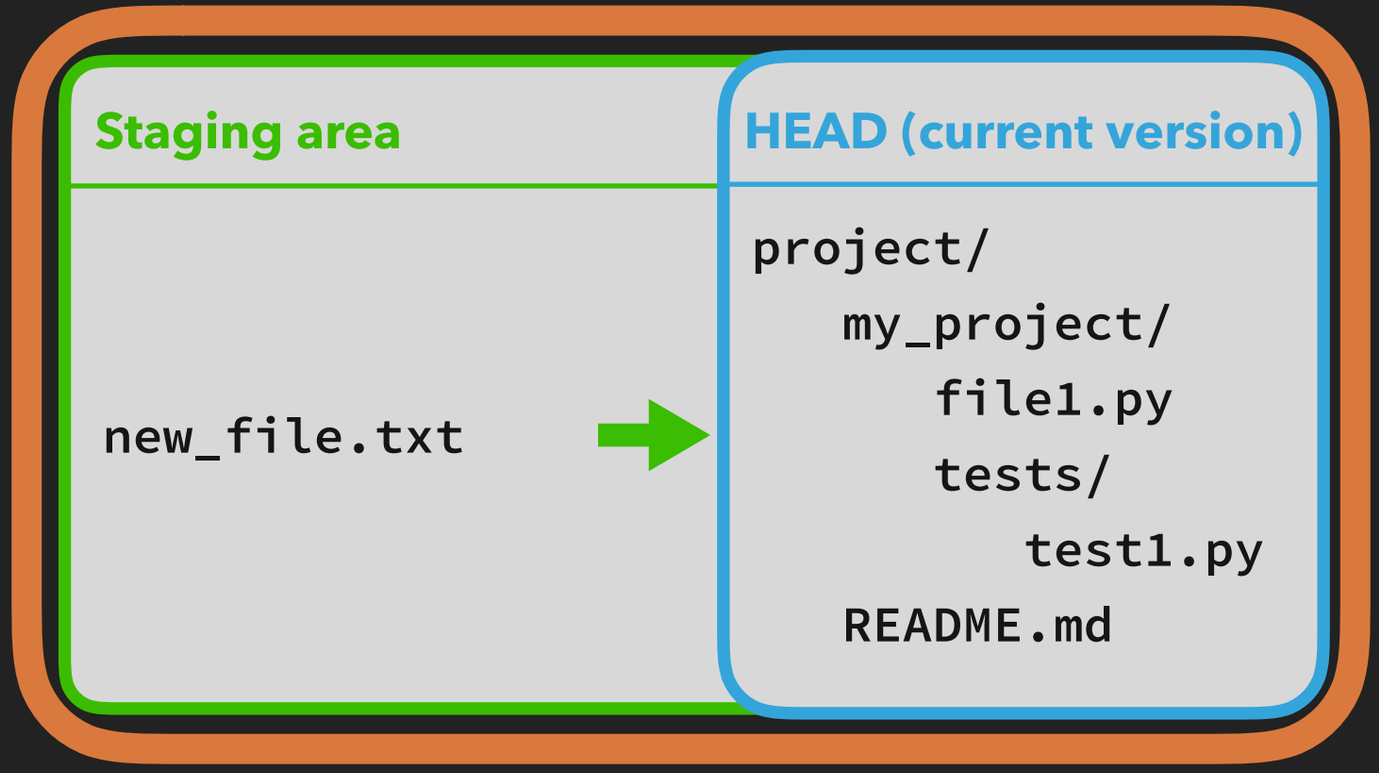
We then have to explicitly add and commit these files to the repository

```
> git add *
> git commit -m “initial commit message”
```

GIT CONCEPTS

Now let's create a new file and stage it

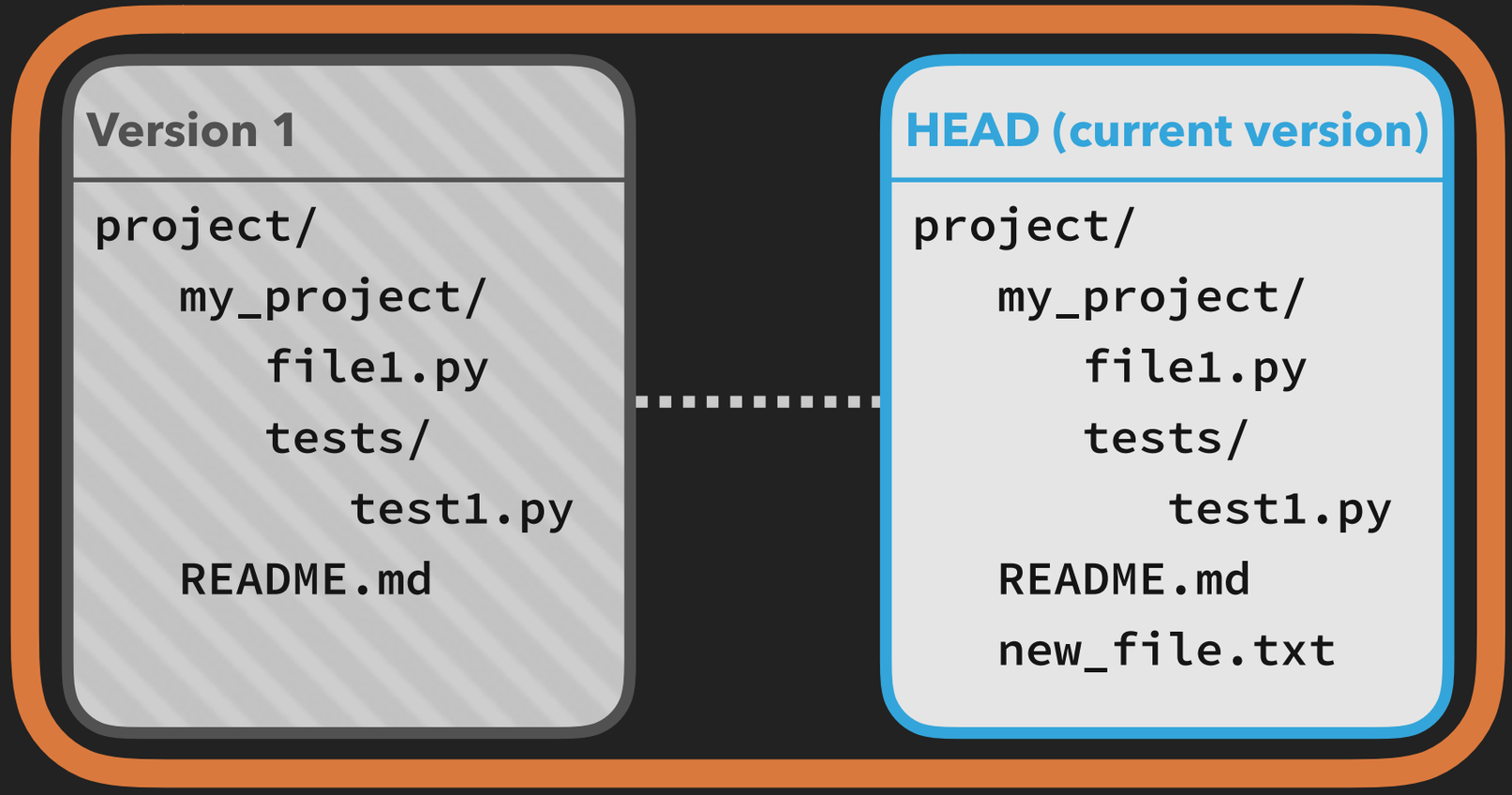
```
> touch new_file.txt
> git add new_file.txt
```



GIT CONCEPTS

Now let's create a new file and stage it

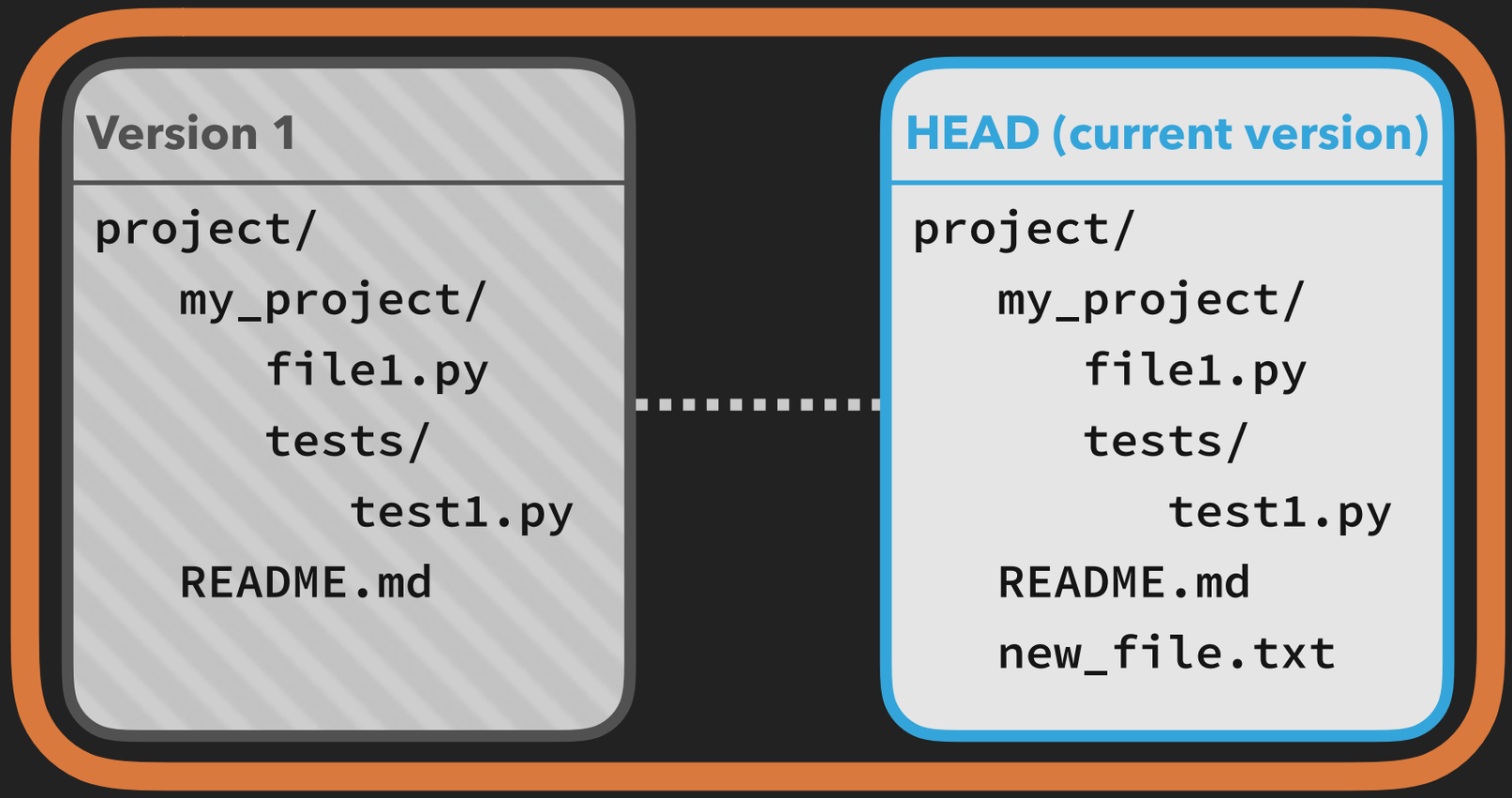
```
> git commit -m "added a new file"
```



GIT CONCEPTS

If we edit the file “`new_file.txt`”, git would notice:

```
> git status
    modified:   new_file.txt
```



GIT CONCEPTS

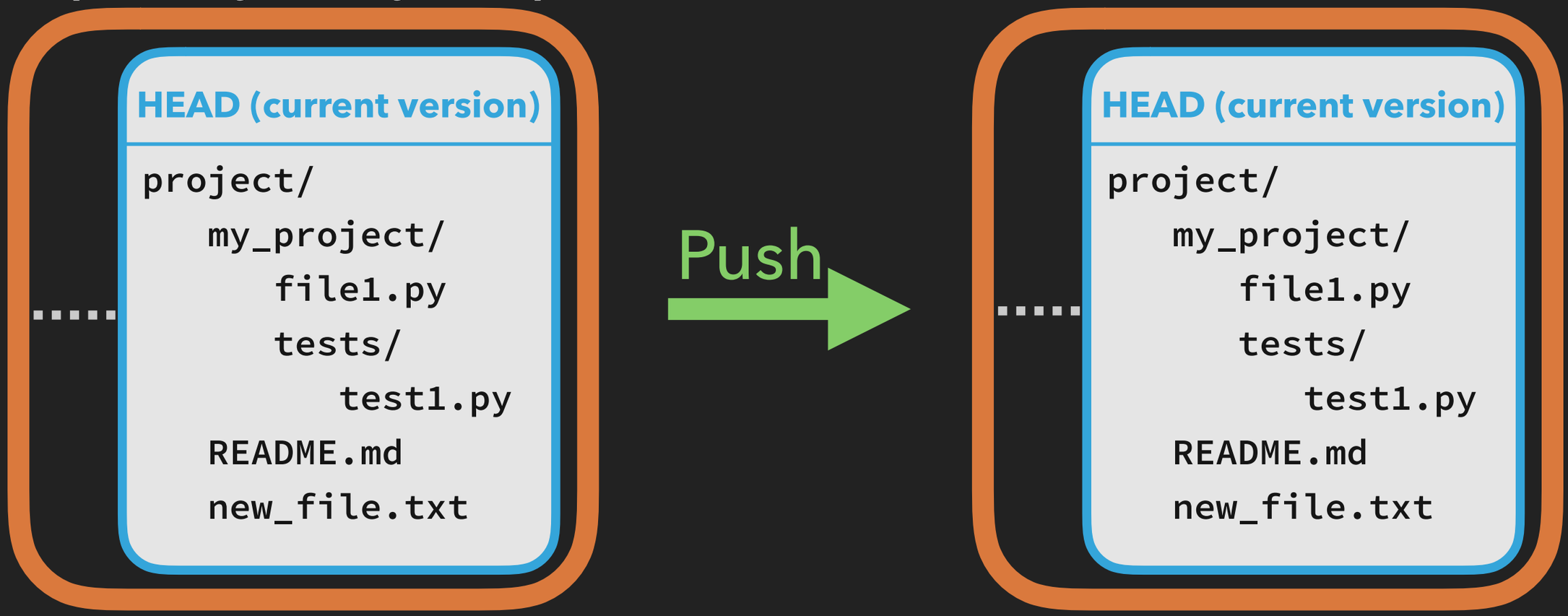
We can then add the changes, and commit them:

```
> git add new_file.txt
> git commit -m “made some changes”
```

GIT CONCEPTS

All of these changes and the repository location are sitting on my computed. What if I want to push this repository to a remote?

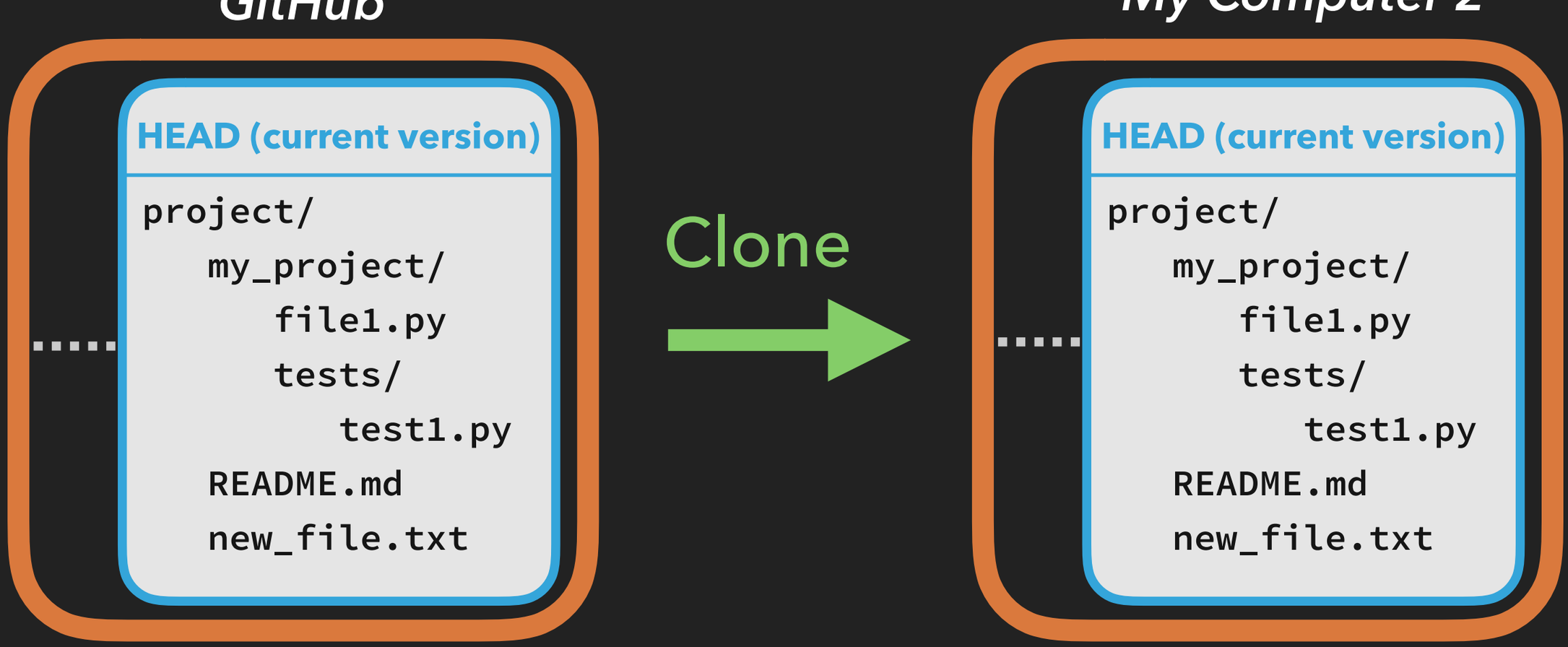
```
> git push
```



GIT CONCEPTS

Now let's say I go to another computer, and I want to ***clone*** the repository over to the new machine

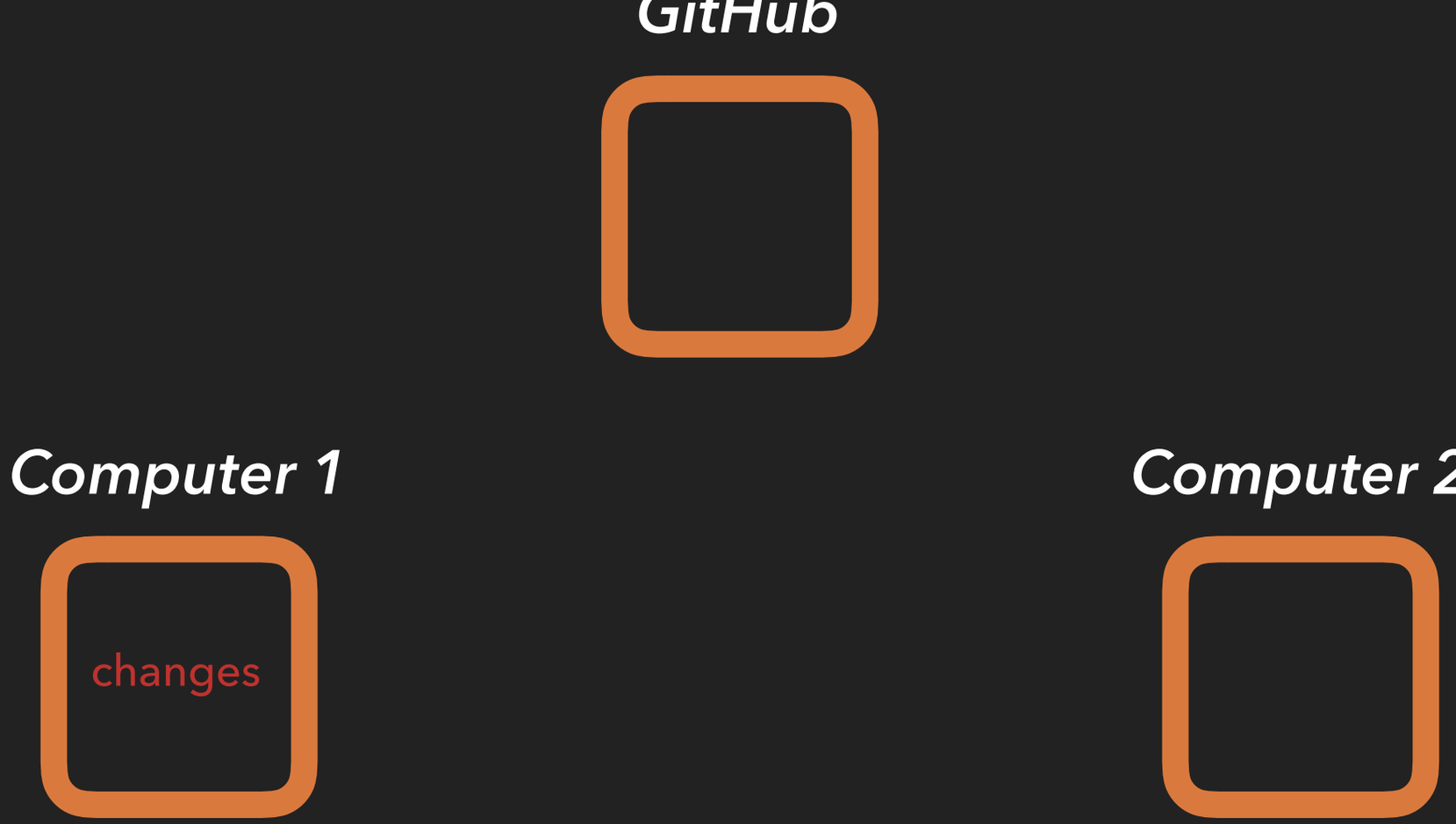
```
> git clone https://github.com/adrn/project
```



GIT CONCEPTS

What if I make a change on computer 1, and want to have those changes over on computer 2?

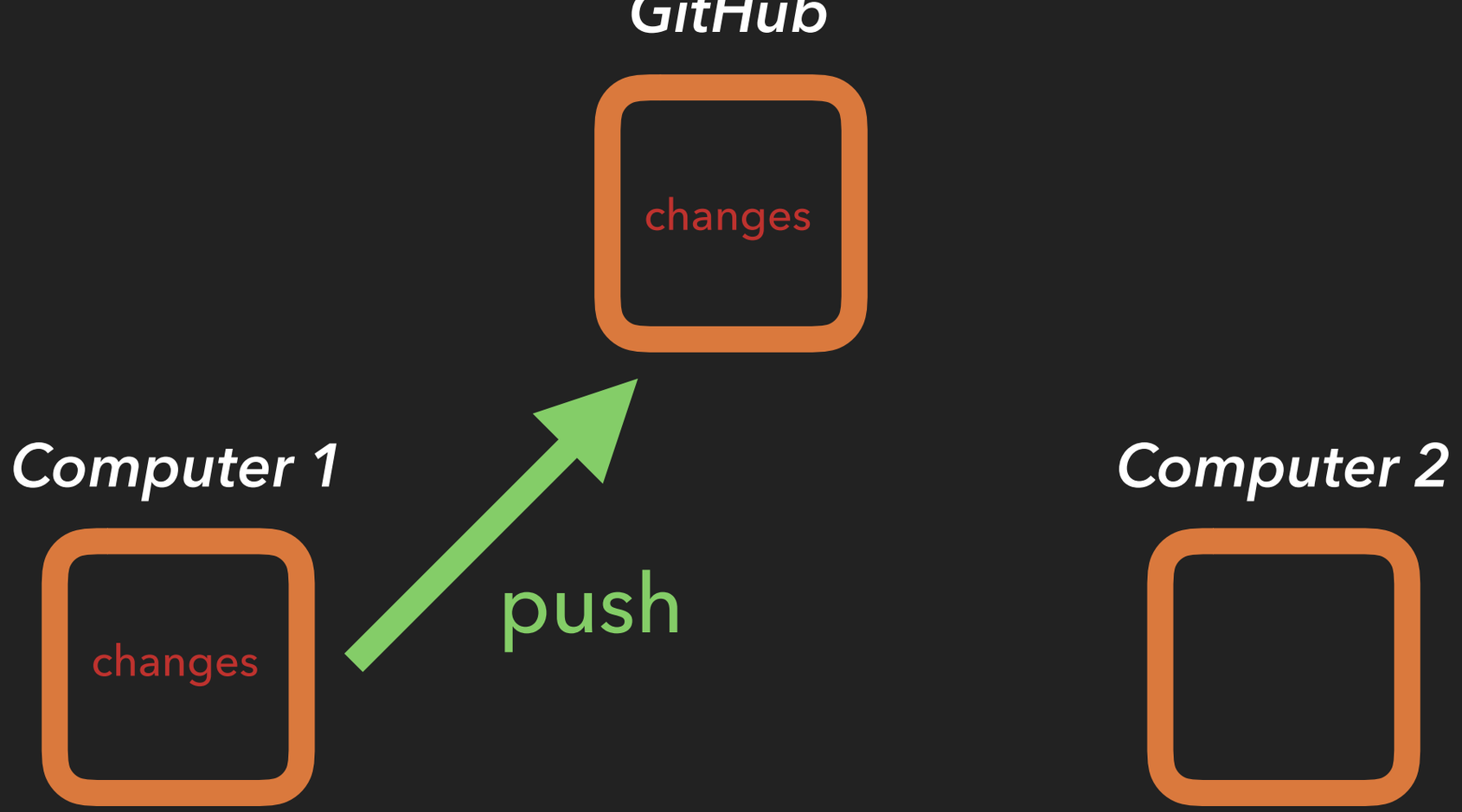
```
on Computer 1:
> git add …
> git commit -m ‘did the thing’
```



GIT CONCEPTS

What if I make a change on computer 1, and want to have those changes over on computer 2?

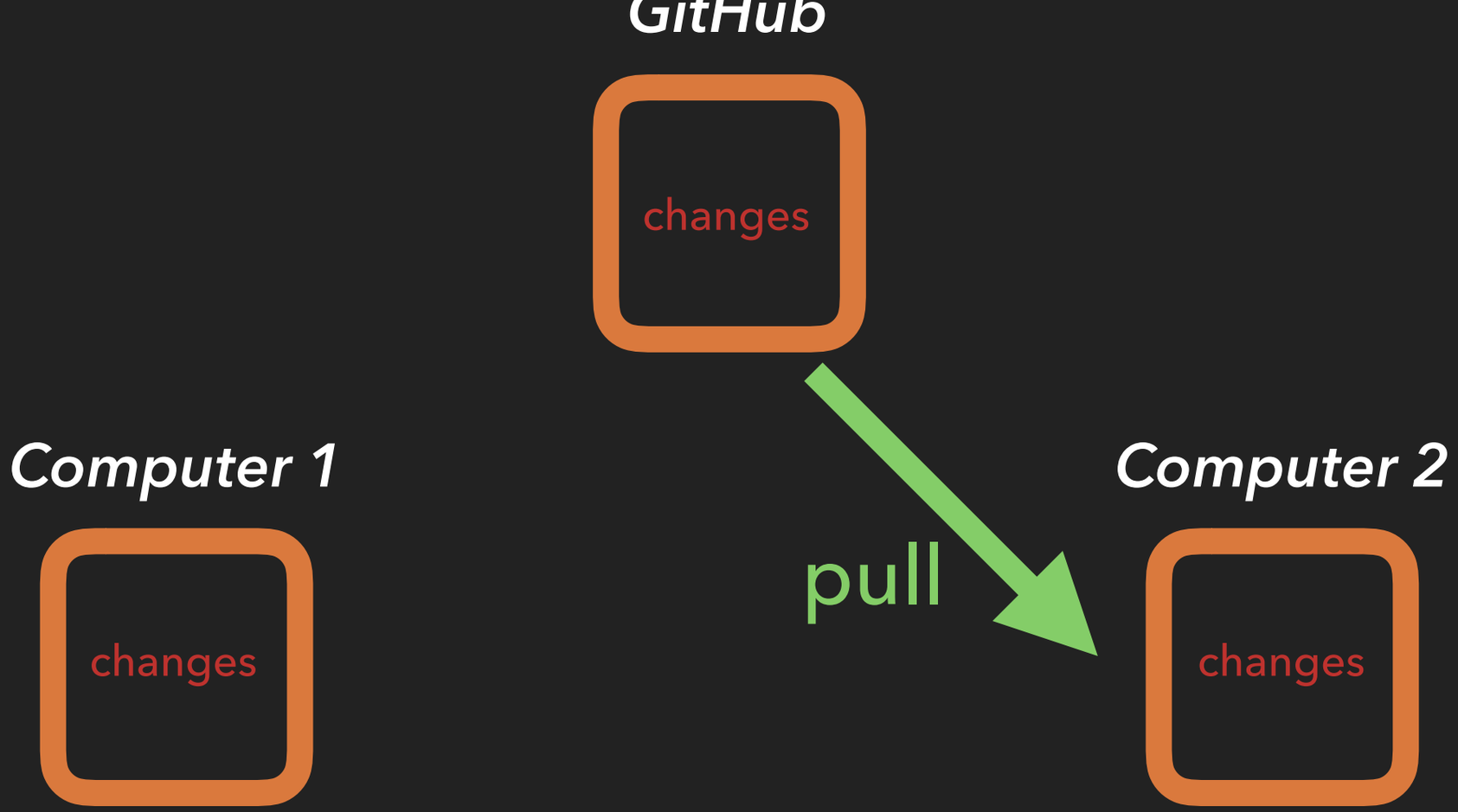
```
on Computer 1:
> git push
```



GIT CONCEPTS

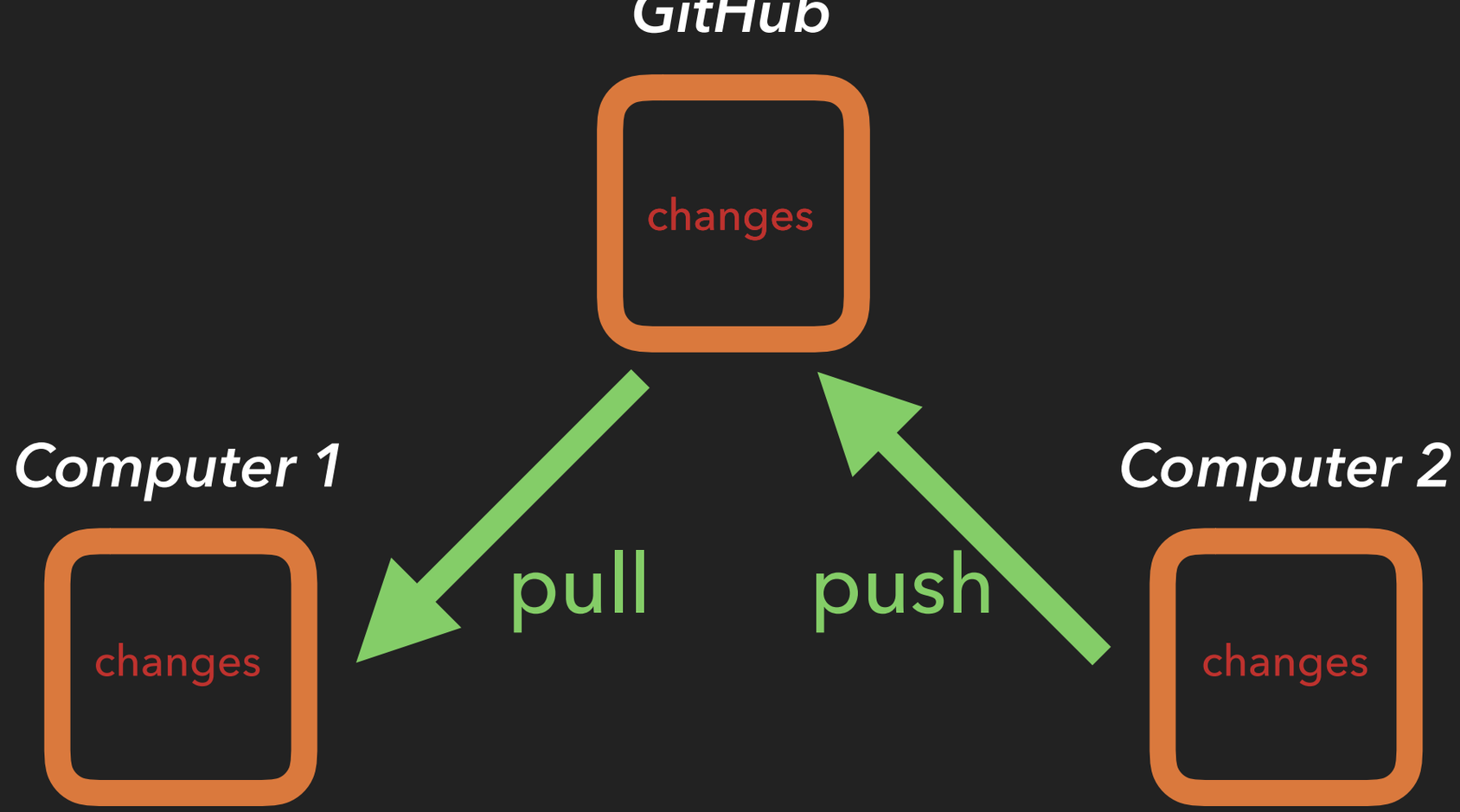
What if I make a change on computer 1, and want to have those changes over on computer 2?

```
on Computer 2:
> git pull
```



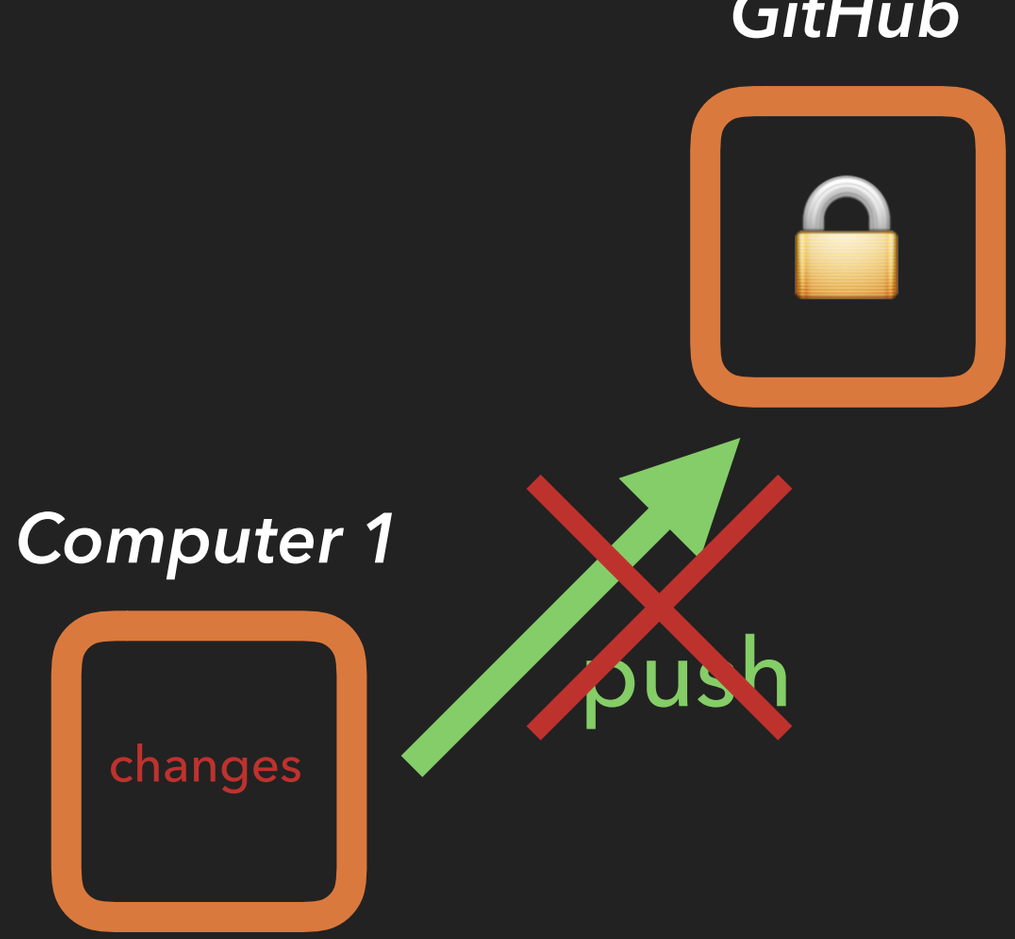
GIT CONCEPTS

It works the same in the other direction from Computer 2 to Computer 1



GIT CONCEPTS

This all assumes that you have ***commit access*** to the repository on GitHub - what if you don't?



GIT CONCEPTS

Instead, you have to submit a ***pull request*** to the repository on GitHub

Co

changes

GIT CONCEPTS

Instead, you have to submit a ***pull request*** to the repository on GitHub