Git Basic Concepts 2

Stat 133 by Gaston Sanchez

Inspecting a Repository

Main git commands to inspect a repo

`git status` & `git log`

About git status

The `git status` command displays the state of the working directory and the staging area. It lets you see:

- which changes have been staged
- which changes haven't been staged
- which files aren't being tracked by Git

About git log

The `git log` command displays information regarding the committed project history.

- Lists the project history
- Filters the project history
- Searches for specific changes

Using git log

```
# lists history of commits
git log

# lists 5 last commits
git log -n 5

# short format listing
git log --oneline
```

Using git log

```
# searches for a particular author
git log --author="gaston"

# searches for a specifc pattern
git log --grep="readme"

# commits from a specific date
git log --since="2018-01-01"
```

Retrieving old versions (go back in time)

You can inspect the contents of a repo at a given point in time.

```
git checkout SHA#
```

```
git checkout 132cb71
git checkout 132cb71 -- file2.txt
```

Undoing Changes

Different Changes

- Changes in the working directory
- Changes in the staging index
- Committed changes

Undoing changes in working directory

You just saved a file (but haven't run *git add* yet) and want to undo the changes:

```
git checkout filename
```

Better to use double-dash:

```
git checkout -- filename
```

(to avoid confusion naming branches)

Undoing a commit

About git revert

There are several ways to “undo” a commit.

The **`git revert`** command is used for undoing changes to a repository’s commit history.

The **`git revert`** command takes a specified commit (SHA-1 number), and inverse the changes from that commit.

Reverting the very last commit

Undo changes in the last commit:

```
git revert HEAD
```

Reverting a commit

Undo changes of a specific commit:

- use the SHA of the old version

```
git revert SHA#
```

(works well for very simple changes)

Amending Commits

The amend option allows you to change the **last** commit (and just the last one):

- either changing a file
- or changing the commit message

```
git commit --amend -m message
```

(amend changes the last commit that HEAD points to)

(it's very difficult to ammend older commits)

git reset to undo many commits (i.e. “rewind”)

Undo many commits by specifying when the HEAD pointer should point to

There are 3 types of reset:

- soft
- mixed
- hard

(use it with extreme caution because it may be very destructive)

Unstaging files

Undoing changes in the staging area:

`git reset HEAD` *`filename`*

(go look at the last commit and reset HEAD to what it was)

(e.g. we don't want to lose the changes, just unstage them)

Reverting a commit

Undo changes of a specific commit:

- use the SHA of the old version

```
git revert SHA#
```

(works well for very simple changes)

Undo reset to undo many commits (i.e. “rewind”)

Undo many commits by specifying when the HEAD pointer should point to

There are 3 types of reset:

- soft
- mixed
- hard

(use it with extreme caution because it may be very destructive)

Soft reset

`git reset --soft SHA#`

- It is the safest of all resets
- does not change staging index
- does not change working directory
- It moves the pointer and does nothing else
- the repo is reset to an earlier version

Mixed reset

`git reset --mixed SHA#`

- It is the default type of reset
- changes staging index to match repository
- does not change working directory

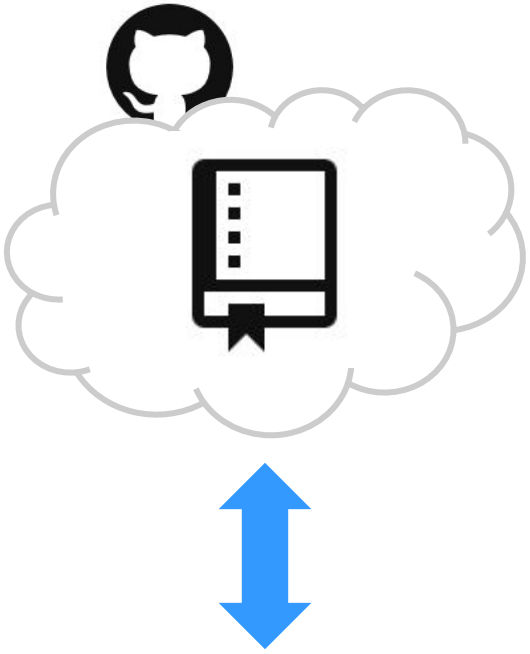
Hard reset

`git reset --hard SHA#`

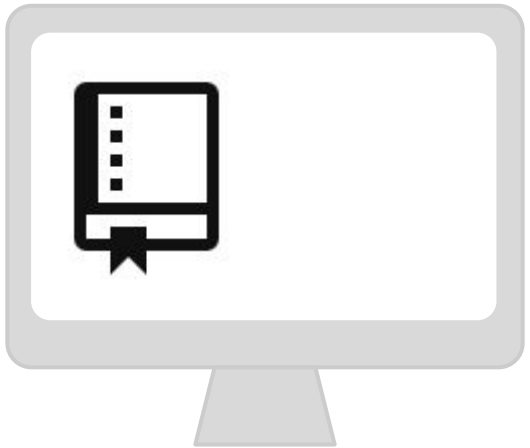
- It is the most destructive type of reset
- changes staging index to match repository
- changes the working directory
- any changes after the specified commit will be obliterated
- use it only for when things went completely wrong and you want to undo the changes

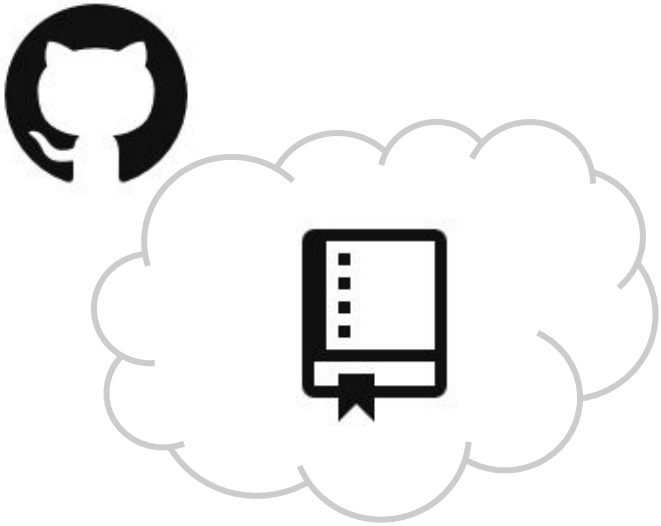
Remote Repositories

remote repository



local repository





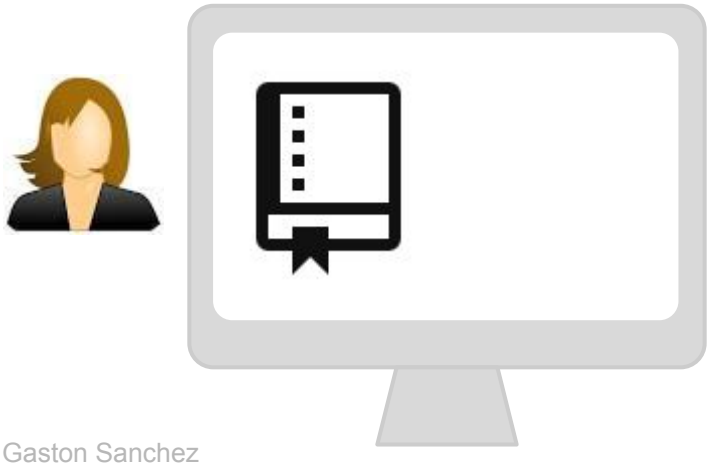
A remote repository needs a place to live

(we'll use GitHub as a home for our remote repositories)

remote repository

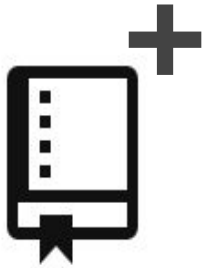
REMOTE REPOSITORIES ARE GREAT FOR COLLABORATION

Jane's local repository



John's local repository

Creating a new repository



I'LL ASSUME YOU HAVE A GITHUB ACCOUNT

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner **Repository name**

 gastonstat /

Great repository names are short and memorable. Need inspiration? How about **garrulous-waffle**.

Description (optional)

Public
Anyone can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** | Add a license: **None**

Create repository

Creating a new remote repository

empty repository

Initialize an empty repo (with nothing in it, no files)
Usually, you use this option to add an existing local repository

vs

minimal repository

Initialize a minimal repository
- with a README file
- maybe a .gitignore
- maybe a LICENSE

Adding a Remote

Once a remote is created, how to establish a connection with it?

What's next?

We need to let our local repository know about the existance of the remote repo.

Remote

```
# list of all the remotes
# that git knows about
git remote
```

`git remote` works a lot like `git branch`

Adding a Remote

```
# adding a remote
git remote add alias url
```

alias is the name of the remote (e.g. origin)

url is the remote's URL

```
# for example
git remote add origin https://github.com/gastonstat/stat133
```

Removing a Remote

```
# deleting a remote
git remote rm alias
```

```
# for example
git remote rm origin
```

Once a remote has been added, you can then start pushing commits to it (and pulling commits from it)

Pushing Commits

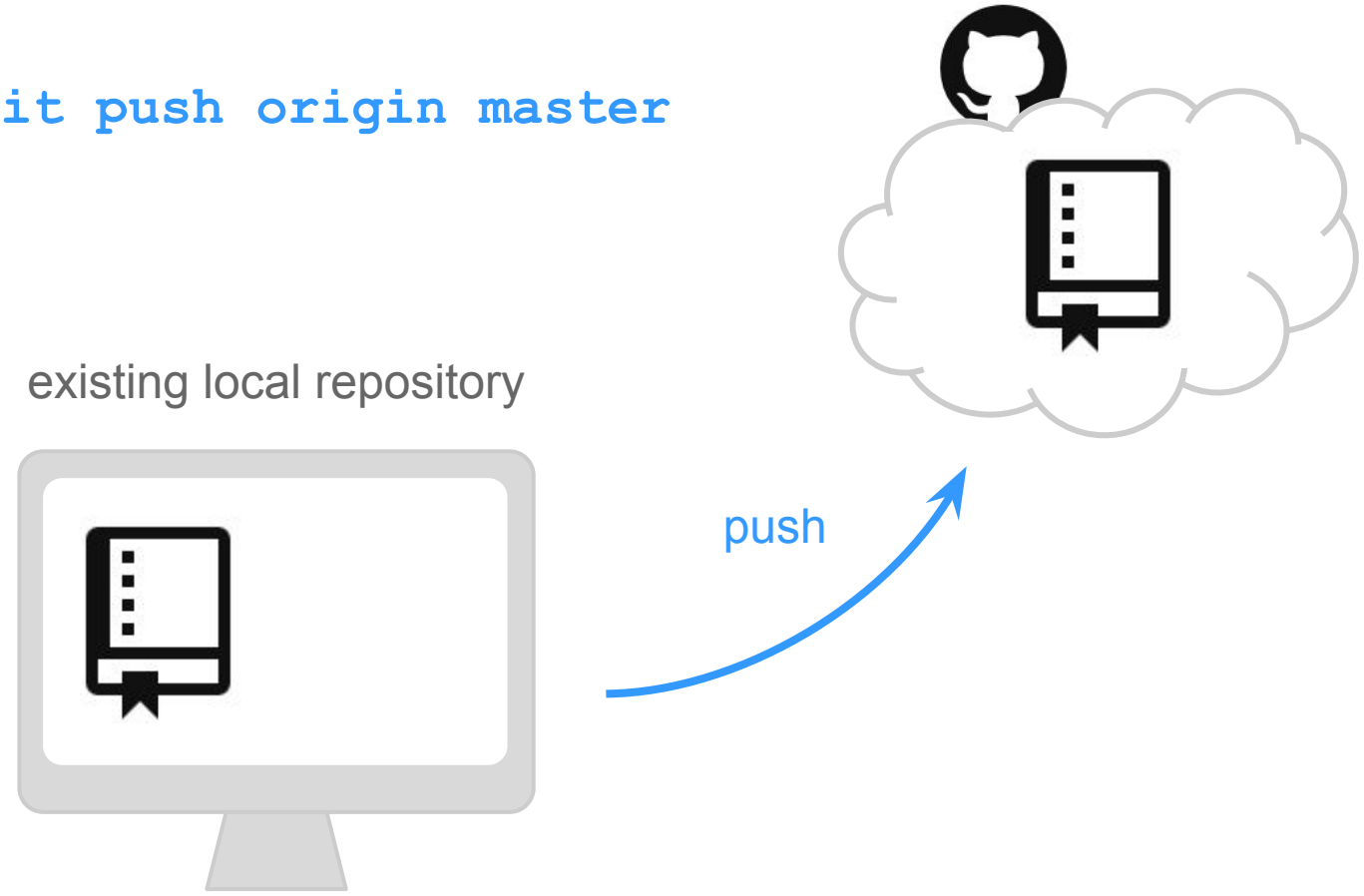
Git push

Putting code in a remote is known as **push**

When we do pushes what we are pushing is a branch

We push to a branch on the remote repository

```
git push origin master
```



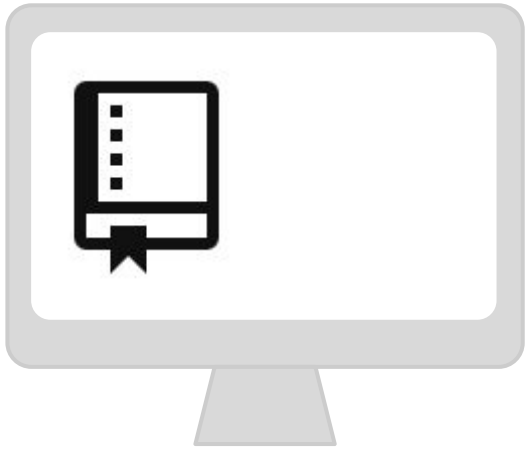
Pushing to a remote repository

```
# git push
git push alias local

# typically we'll use this:
git push origin master
```

Pulling Commits

pull



`git pull origin master`

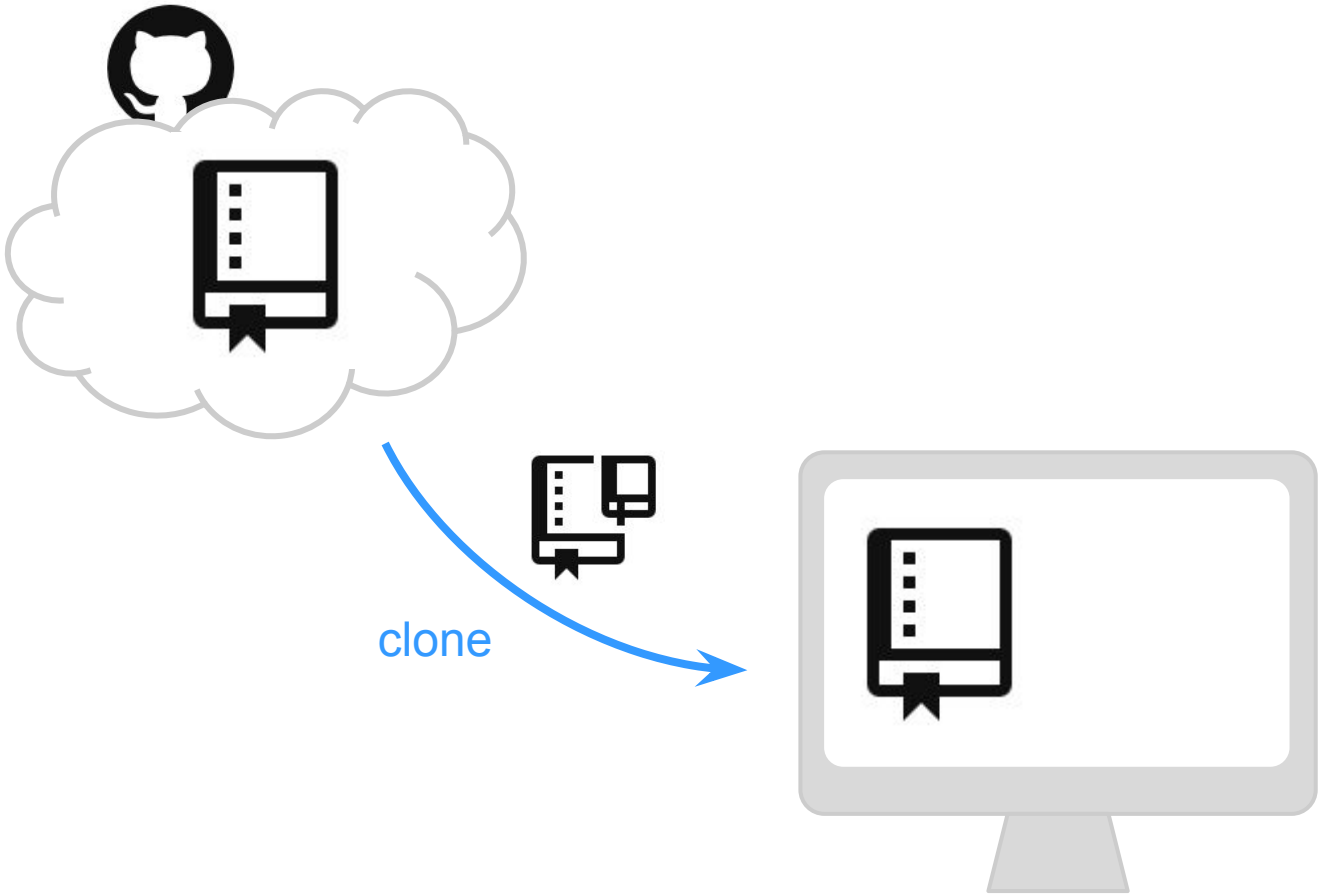
Pulling from a remote repository

```
# git pull
git pull alias local

# typically we'll use this:
git pull -u origin master
```

Cloning a Remote

Cloning a remote repository




```
git clone https://github.com/username/repository.git
```

About cloning

Instead of taking a repository that's local and pushing it to the remote, it's important to know how to do the reverse.

In other words, you must learn how to take a remote repository and pull it down to have a local copy to work with it (assuming you don't have a local copy to star with)

Cloning a repository

```
# clone
git clone repo_url

# for example
git clone https://github.com/BIDS/2016-01-14-berkeley.git
```

Cloning a repository

```
# for example
git clone https://github.com/BIDS/2016-01-14-berkeley.git
```

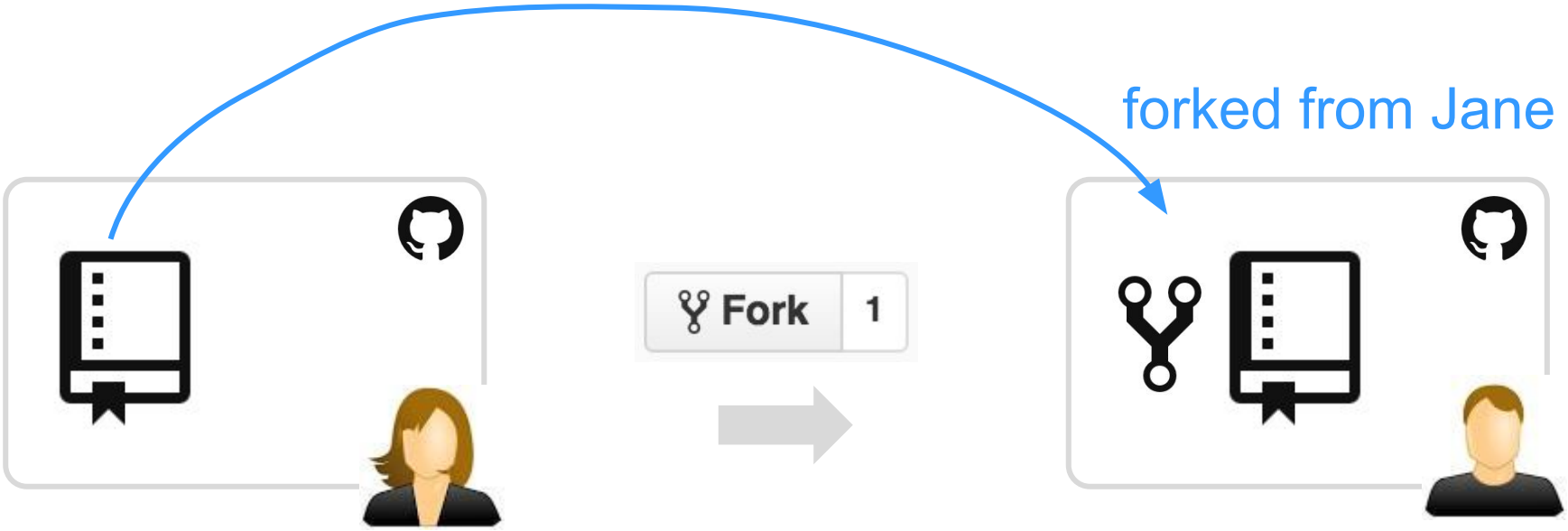
by default **git clone** creates a local repository with default name as the remote, in this case “2016-01-14-berkeley”

however, you can give a different name:

`git clone` *`repo_url new_name`*

Forking a Remote

Forking an existing repository



You can obtain a copy of someone else's repository by **forking** that repository in your GitHub

About Forks

A fork is a copy of a repository

Forking a repo allows you to experiment without affecting the original project

Commonly used to propose changes

Use someone else's project as a starting point

forked from jane