Intro to Git

Why Version Control?

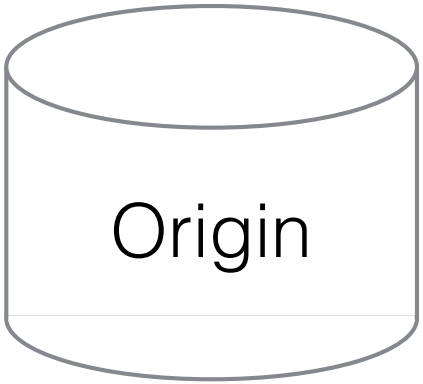
- **Backup and restore** - because accidents happen
- **Synchronization** - multiple people can make changes
- **Short term undo** - that last change made things worse?
- **Long term undo** - find out when a bug was introduced
- **Track changes** - all changes related to a bug fix
- **Sandboxing** - try something out without messing up the main code
- **Branching and merging** - (better defined sandboxes)

https://betterexplained.com/articles/a-visual-guide-to-version-control/

Git

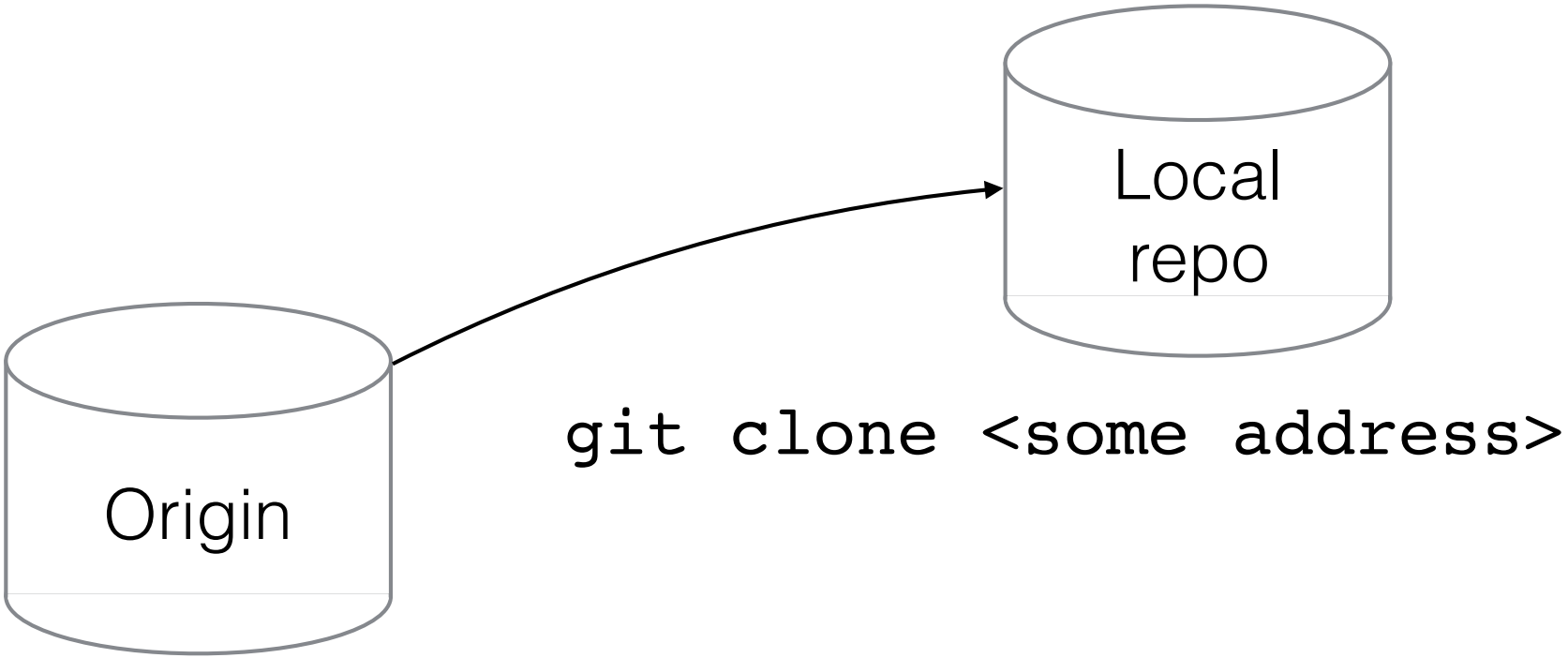
- For CSC209, we will be creating repositories for you.
- These repositories live on a department server
- You will clone your remote repository, work on files locally, add and commit changes to your local repository, and push changes to the remote repository

Remote repository



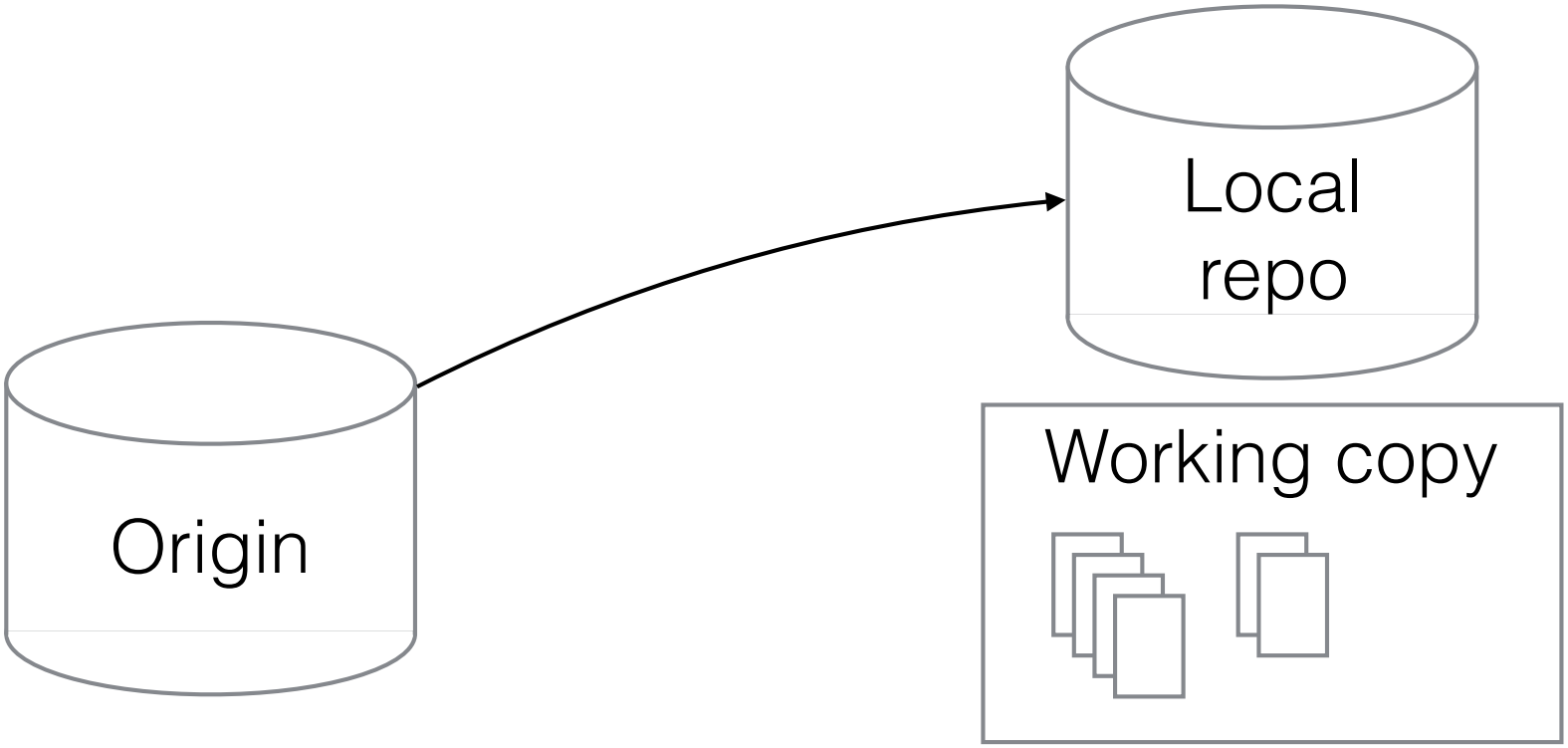
- This is the repo that lives on another server.
- By convention we call it the origin
- In Github terminology, you may have an “upstream” repo and a forked copy of the repo called the “origin”, but we are working with a simpler model.

Clone to get local repo



- The “clone” command makes a copy of the remote repository on your local machine.
- Now there are two repositories. (Git is a distributed version control system)

Clone to get local repo

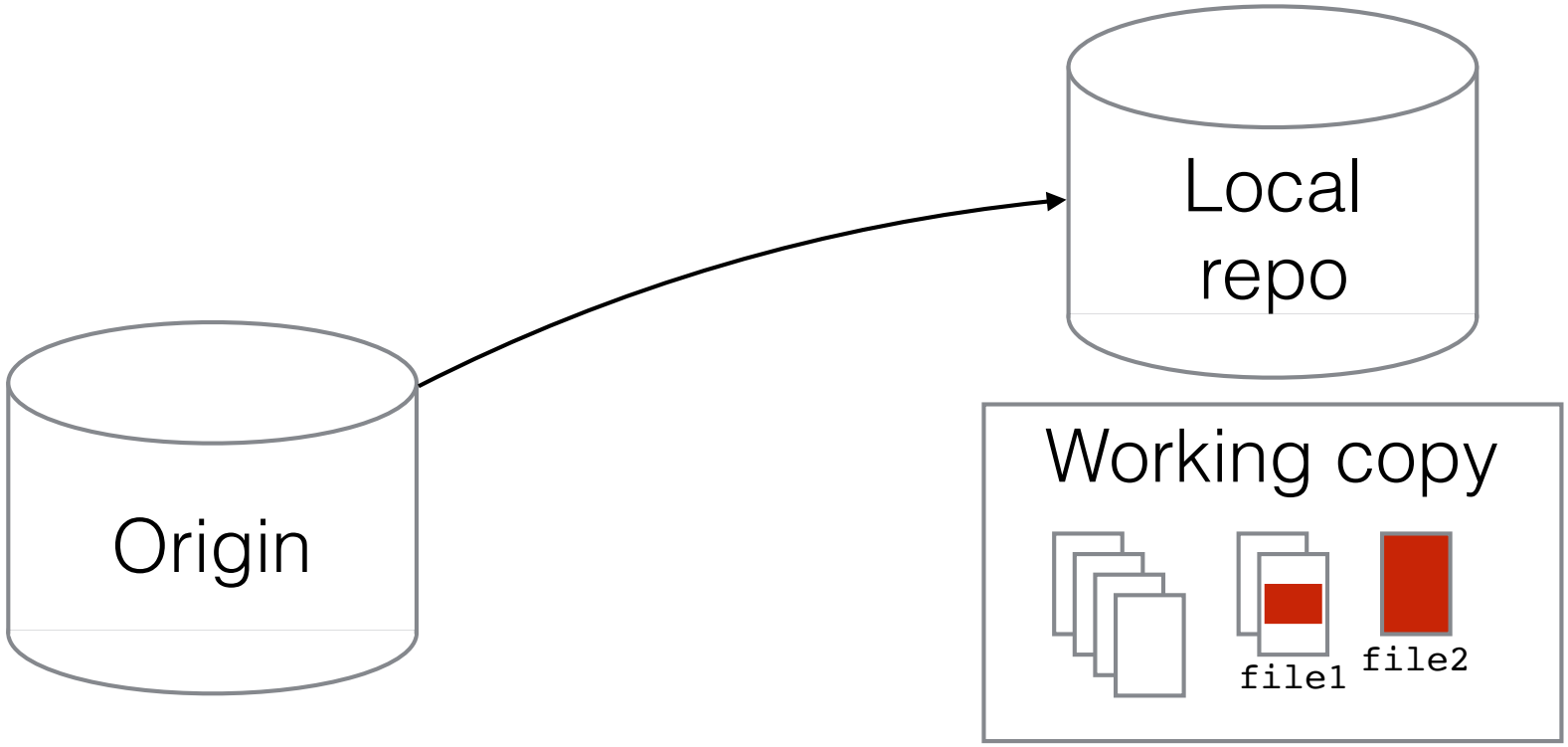


- The “clone” command makes a copy of the remote repository on your local machine.
- Now there are two repositories. (Git is a distributed version control system)

Local repository

- The actual repository is hidden. What you see is the working copy of the files from the repository.
- Now you can create new files, make changes to files, and delete files.
- When you want to “commit” a change to the local repository, you need to first “stage” the changes”

How to get work done



- Make changes to files, add new files. When you are ready to commit your work to your local repo you need to tell git which files have changes that you want to add this time.

```
git add file1 file2
git commit -m “adding feature x”
```

Staging changes

- **`git add`** doesn't add files to your repo it prepares changes to add to a commit.
- This means that when you make some changes to a file and then add and commit them, the next time you make some changes to a file you will still have to run **`git add`** to add the changes to the next commit. This is different than subversion.

git status

- A file can be in one of 4 states:
  - **untracked** - you have never run a git command on the file
  - **tracked** - committed
  - **staged** - `git add` has been used to add changes in this file to the next commit
  - **dirty/modified** - the file has changes that haven't been staged

- TIP: Use `git status` *regularly*. It helps you make sure the changes you have made really make it into the repo.

Typical work flow

- Starting a project:
  - `git clone <url>`
  - `git remote add origin <url>`
- Normal work:
  - After you have made some changes
  - `git status` (see what has really changed)
  - `git add file1 file2 file3`
  - `git commit -m “meaningful commit message”`
  - `git push`

Typical work flow

- Starting a project:
  - `git clone <url>`
  - `git remote add origin <url>`
- Normal work:
  - After you have made some changes
  - `git status` (see what has really changed)
  - `git add file1 file2 file3`
  - `git commit -m “meaningful commit message”`
  - `git push`

Save changes to local repo

Push changes to the remote repository.