Using Git/Github

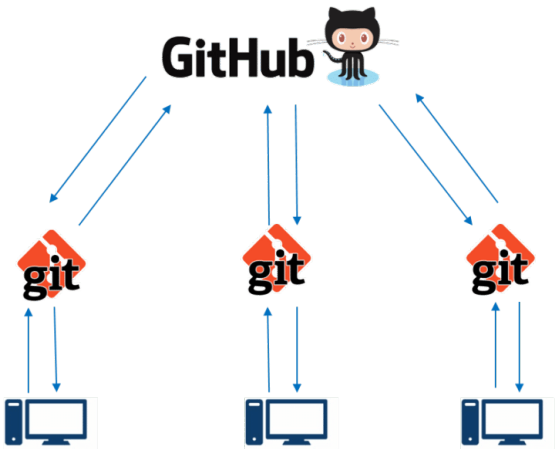
Barum Park (b.park@cornell.edu)
Department of Sociology
Cornell University

What is Git?

Git is a *version control system* that helps you keep track of your code.

It's widely used among software developers and scientists to collaborate on projects.

How it Works



Source:
`https://www.edureka.co/`

Github is the server where you save multiple repositories (“repo”s)

You can think of *Github* as a (potentially shared) “dropbox” to store your code, where repositories are folders/directories

You use *Git* to communicate with Github:

- After working on your code, you *push* your changes to Github
- Other collaborates then *pull* your changes into their local machines

How it Works

Git/Github makes sure that the *entire history* of all changes are recorded

- You can always go back in time to any version of your repo
- No more, `code.R`, `codeV1.R`, `codeFinal.R`, `codeFinalV1.R`, `...`
- Your collaborators can see & comment on the changes you've made

The Command Line

The Command Line

To use Git, you have to become familiar with the command line

- On MacOS, search for the terminal via ⌘ + Space followed by `terminal`
- On Windows, open `Git Bash`

You should see something like the following:

Basic Commands

Here is a list of basic commands that we'll use:

1. `pwd` : this prints the path of the current directory
2. `ls` : lists files/directories in current directory
3. `mkdir <path>` : creates a directory in `<path>`
4. `cd <path>` : changes the directory to `<path>`
5. `touch <filename>` : creates a file named `<filename>`

Let's try these out in order ...

Short Exercise: Getting Ready

Do the following **in order**:

1. Open your terminal
2. type `pwd` to check your location
3. type `ls` to see what files you have in the current directory
4. type `mkdir git_tutorial` to create a new directory named `git_tutorial`
5. type `ls` to check that the directory was created
6. move into this directory by typing `cd git_tutorial`
7. type `pwd` to check that you are in the right directory
8. create a new file using `touch myfile`
9. type `ls` to check that the file was created

Short Exercise: Getting Ready

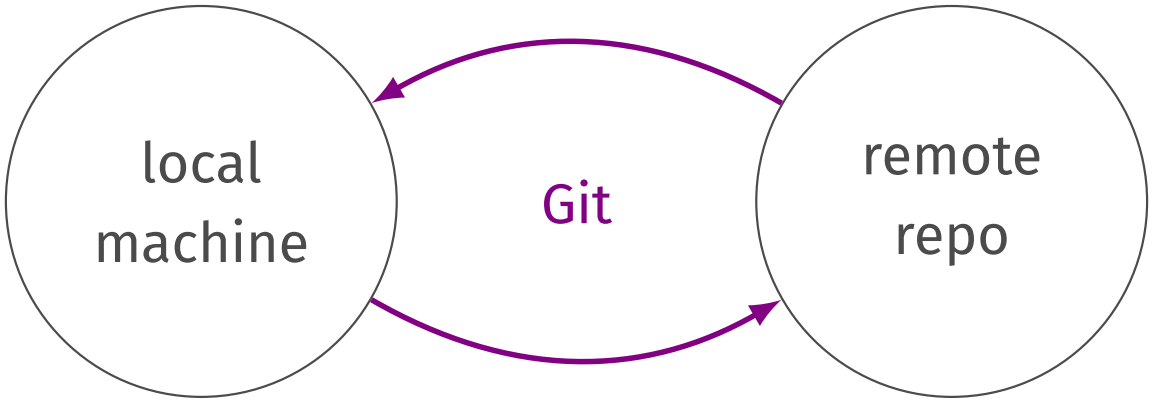
If everything was done successfully,

- You'd have created the directory `$HOME/git_tutorial`
- You are currently located in this directory
- The directory contains one file, namely `myfile`

Keep the terminal open (we'll use it later on ...)

Setting up Git/Github

Overview



Our goal for this section will be

1. Creating a *remote* repo on Github
2. Initializing Git on a *local* directory
3. Link our local git directoroy to the remote repo

Creating a Repo on Github

To create a repo on Github:

1. Go to `https://github.com/` and log in to your account
2. Click on the 

button on the left (this will bring you to a new window)
3. Enter the repository name as `testrepo`
4. Leave everything else as is

This will create a repo named `testrepo` on your Github!

You can think of this repo as an empty directory that we are going to fill

Don't close this window (we'll need it later)

Creating a local repo

After creating a (remote) repo on Github, there are two ways to link a local repo (i.e., directory) to the remote repo

We can either

1. *clone* the remote repo into our local computer;
2. or use a directory that we have created and link it to the remote repo

As we have already created a directory using the terminal, let us consider the second method in this tutorial ...

Initializing

To link a local directory to the remote repo we've created, we need to *initialize* a local git repo

In the terminal,

1. use `pwd` to check that you are in the directory that you want to link
2. Then type `git init` to initizlize the git repository
3. Check with `ls` what has changed in the directory (you should see only `myfile` in your directory)

It seems that nothing has changed, but when you type `ls -a`, you'll see that a directory named `.git` was created.

If so, you've done everything correctly.

Linking

With this, your directory is a local Git repo (!)

It remains to *link* this local repo to the remote repo we've created on Github

This can be done with

```
git remote add origin <remote address>
```

It remains to find out what the address of our remote repo is, alas.

When you return to the browser, you'll see that the address of your remote repo is stated there

(It should be `https://github.com/<user-name>/testrepo.git`)

Linking

So, what you should type into your terminal is

```
git remote add origin https://github.com/<user-name>/testrepo.git
```

with `<user-name>` substituted with your Github account name.

Here's an explanation of the code:

- `git remote add` means that we want to link a remote repo to the local repo
- `origin` is what we want to call the remote repo (we use "origin" as the name, following convention)
- `<address>` is the address of the remote repo

With this our local repo and our remote repo are linked!

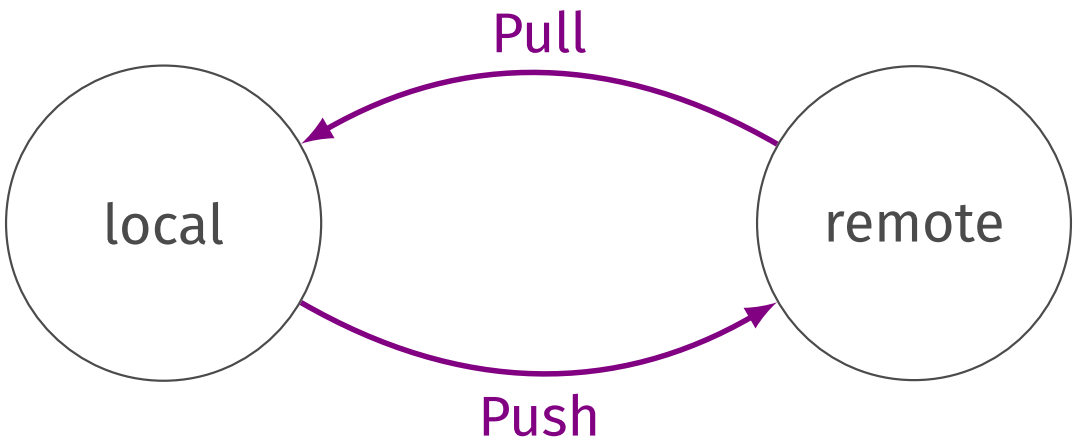
Add, Commit, Push, and Pull

Overview

Once our local and remote repos are linked, we are able to update either of these with the other

Updating the remote repo based on the local repo is called *pushing*

Updating the local repo based on the remote repo is called *pulling*



From Local to Remote

There are three steps involved when we want to update the remote based on the local repo

1. We need to let Git know what files to track (*add*ing)
2. *commit* the change we make on the tracked files, after which
3. we *push* them upstream

Add, Commit, and Push

1. To tell Git to track a file, we use the `git add` command
   In your terminal, you can make Git aware of the `myfile` by typing `git add myfile` (A very useful shortcut to "add" all files in the current directory is to type `git add .`)

2. Thereafter, we "commit" these changes by typing

   ```
   git commit -m "start tracking myfile"
   ```

   The `-m "<text>"` is required and intended to leave a short note for every change we commit

3. Lastly, we push these changes to our remote repo with

   ```
   git push origin main
   ```

   where `main` is the "branch" to which we want to push (more on this later)

Pulling

When you refresh your browser window, you'll see that the `myfile` file has appeared there.

When you click on the file, Github will show "nothing" (since the file is empty)

1. Let's edit this file a bit by clicking on the edit () button.
2. Write something into this file (e.g., "hello world")
3. Click on "commit changes" at the bottom (keep "Commit directly to the main branch" selected)
4. Check that your edits are reflect on the remote repo

Pulling

To incorporate these edits in your local repo as well, switch to the terminal and type

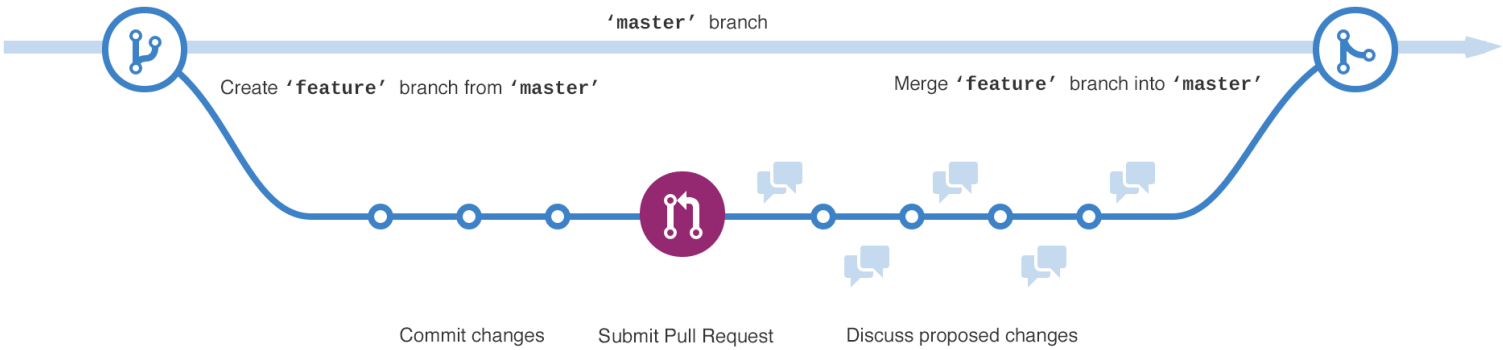
```
git pull origin main
```

where **`git pull`** tells Git that we want to pull from the remote and **`origin main`** means that we want to pull from the “main” branch of the remote named “origin.”

When you open `myfile`, you’ll see that the empty file has now some text in it!

Branching

Branching



You can think of a **branch** as a "version" of your git repo

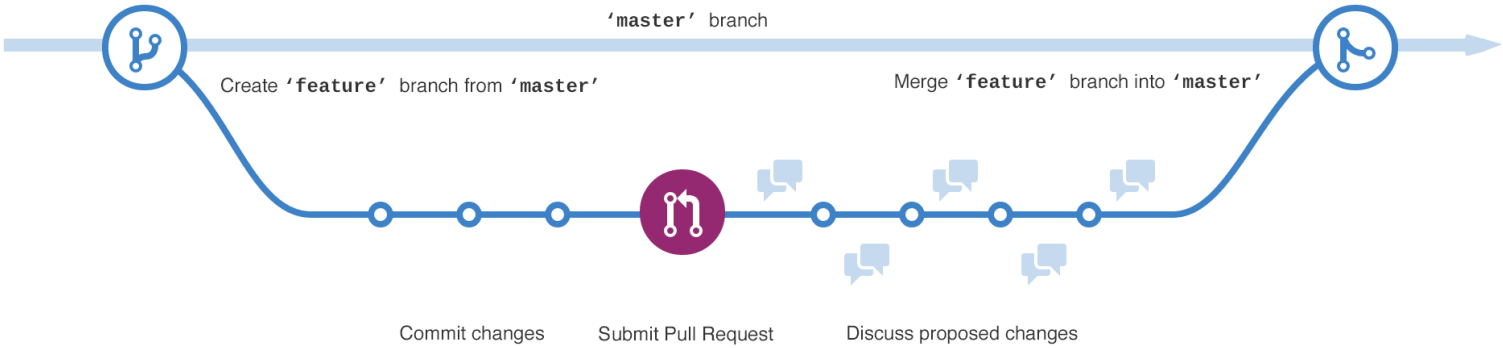
By creating multiple branches, Git allows you to keep multiple versions of your repository

The default branch on Git/Github is the `main` branch

(this is why we typed `git push origin main`)

It is a good idea to keep the default branch "stable"—i.e., you should be always sure that your code on the `main` branch is working without error

Workflow with Branching



The usual workflow with Git/Github is as follows:

1. You want to add stuff to your stable code on the `main` branch
2. Create a new branch, say we call it `new`
3. Work on the `new` branch by adding code
4. Create a `pull request` to merge the added changes in `new` to `main`
5. After reviewing the pull request and check that everything works, you merge `new` into `main`

On What Branch am I?

To check on which branch you are working, you can use

```
git status
```

When you type it now, you'll see something like the following:

```
On branch main
nothing to commit, working tree clean
```

This tells you that you are working on the `main` branch

To create and switch to a new branch named `test`, we can us

```
git checkout -B test
```

where the `-B` option indicates that we want to create a new branch.

After running that command, check your branch with `git status` again

Working on the Test Branch

Let us create a new file, this time with the `echo` and `>` command line codes:

```
echo "working on the test branch" > myfile2
```

We can check that the file was created using the `ls` command. To check the content of both `myfile` and `myfile2` with the `cat` command:

```
>>> ls
myfile  myfile2
>>> cat myfile
hello world
>>> cat myfile2
working on the test branch
```

Pushing to a New Branch

As we have created something new, we add, commit, and push these change

```
>>> git add .
>>> git commit -m "creating my second file"
>>> git push origin test
```

Notice that we are `pushing` to the `test` branch on the remote `origin` (not `main`)

Since there is no `test` branch on the remote, Github will automatically create it there

When you refresh your browser, you'll see the message " test had recent pushes less than a minute ago," which indicates that the branch was created

You can check this also by clicking on the 

button

What Happened to the Main Branch?

The work we've done on the `test` branch leaves the `main` branch (our default branch) unchanged

We can check that this is indeed the case

```
>>> git status
On branch test
nothing to commit, working tree clean
>>> ls
myfile  myfile2
>>> git checkout main
Switched to branch 'main'
>>> ls
myfile
```

Notice that we've not used the `-B` option, since we don't want to create a new branch

Pull Requests

After we've checked that there is no errors in our new work (i.e., `myfile2`), we are ready to merge these changes into our stable `main` branch

When we look into the browser, we see that a Compare & pull request button has appeared after our last push to the `test` branch.

Let's click it, which will starts the process of creating a `pull request`

Pull Requests

Github will immediately determine whether the two branches have any conflicts. If there are none, it will tell you that these branches are ✓ Able to merge..

In the next slot, we can add some comments describing what changes this pull request contains

In the last part of the window, Github compares the `main` and `test` branch and will summarize

1. all commits that were made on the branch; and
2. all changes that were made to new or existing files

If we are happy with these changes (which we are!), we click on the Create pull request button to create the `pull request`

Merging

With this, the pull request is created.

When you are working alone on this repo, you can go on and `merge` the pull request into the `main` branch.

When you are collaborating with others, you can ask your collaborators to review the changes made in this pull request by adding them via the “Reviewer” tab

The section at the bottom of the page is also useful for collaborators/reviewers in order to ask you to make further changes or raise issues

As we have no collaborators (☹), let’s go ahead and merge the branches by pusing on Merge pull request and then confirming the merge

Pulling Changes

While the two branches were merged on Github, these changes have not yet been incorporated into our local repo. To do this we need to `pull` from the remote.
First, we check whether there are any uncommited changes

```
>>> git status
On branch main
nothing to commit, working tree clean
```

Then, we pull from the `main` branch (as the `test` branch was merged into `main`)

```
>>> git pull origin main
remote: Enumerating objects: 1, done.
...
Fast-forward
myfile2 | 1 +
1 file changed, 1 insertion(+)
create mode 100644 myfile2
```

Checking Changes

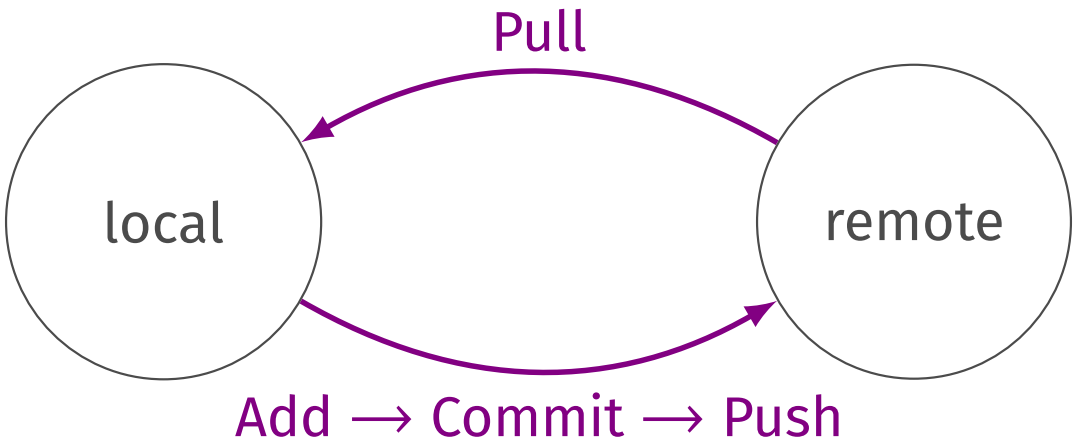
Lastly, we might check whether the changes were successfully incorporated

```
>>> ls
myfile  myfile2
```

We see that `myfile2`, which was on the `test` branch but not on `main` before the merge, appeared on the main branch as well!

Wrapping Up

Summary



What we've discussed today was mainly about how to keep the *local* and *remote* repositories in sync via pushing and pulling

We've discussed how to create branches as well

While these two functions of Git/Github are certainly imporant, they are only the tip of the iceberg of what you can do with these tools

Further Topics

Other topics of importance are

- `Forking`: copying other developers' repos into your Github (to contribute to them)
- `Cloning`: copying other developers' repos into your local machine
- `Going back in time`: How to revert your git repository to a previous time point (in case somethiing goes wrong)
- `Markdown`: The typesetting language used throughout Github, which is important to communicate with others
- `Collaboration`: How to collaborate on a git repository with others, which includes raising *issues*, *code reviews*, etc.

Final words

Using Git/Github is great to streamline your workflow and to collaborate with others on code

Regardless of whether you are collaborating or working alone, keeping the following in mind will be helpful

- don't forget to *commit*: commiting more is better than less
- when committing, use informative commit messages
- Always keep your default branch stable, most of the work is done on separate branches
- Be patient ...

GOOD LUCK ☺