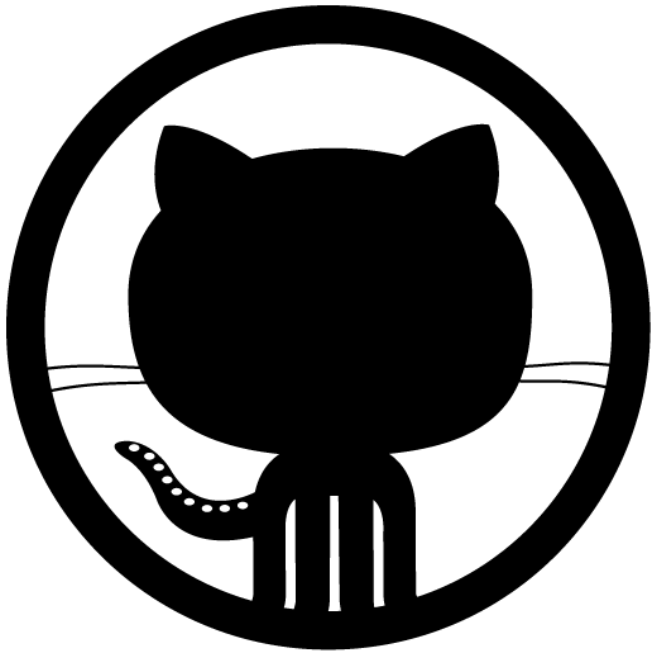
GENERAL ASSEMBLY

DATA SCIENCE

SYD DAT 8

Week 1 Lesson 2 - Git
Thursday 25th May 2017

GITHUB



WHY LEARN GIT (OR ANY VERSION CONTROL)?

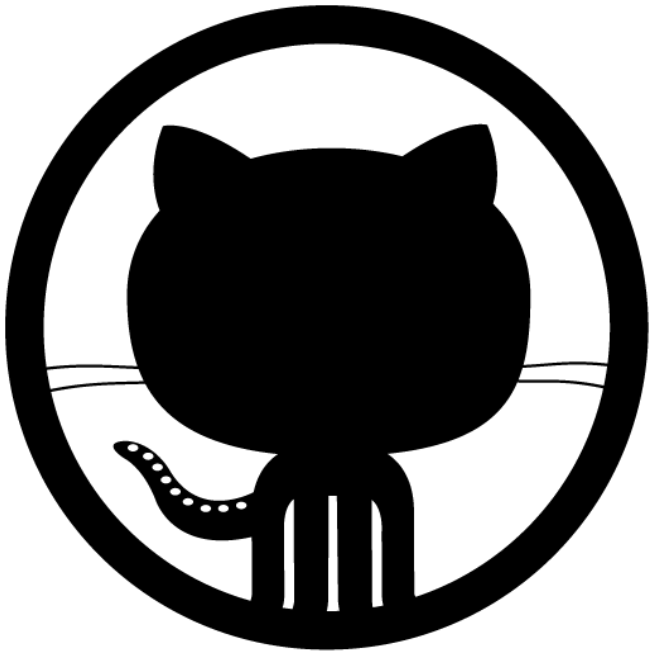
- Version control is useful when you write code, and data scientists write code
- Enables teams to easily collaborate on the same codebase
- Enables you to contribute to open source projects
- Attractive skill for employment

WHAT IS GIT?

- Version control system that allows you to track files and file changes in a repository (“repo”)
- Primarily used by software developers
- Most widely used version control system
- Alternatives: Mercurial, Subversion, CVS
- Runs from the command line (usually)
- Can be used alone or in a team

WHAT IS GITHUB? 5

- Allows you to put your Git repos online
- Largest code host in the world
- Alternative: Bitbucket
- Benefits of GitHub:
  - Backup of files
  - Visual interface for navigating repos
  - Makes repo collaboration easy



Git does not require GitHub

- Designed (by programmers) for power and flexibility over simplicity
- Hard to know if what you did was right
- Hard to explore since most actions are “permanent” (in a sense) and can have serious consequences
- We’ll focus on the most important 10% of Git

- Links to more resources:

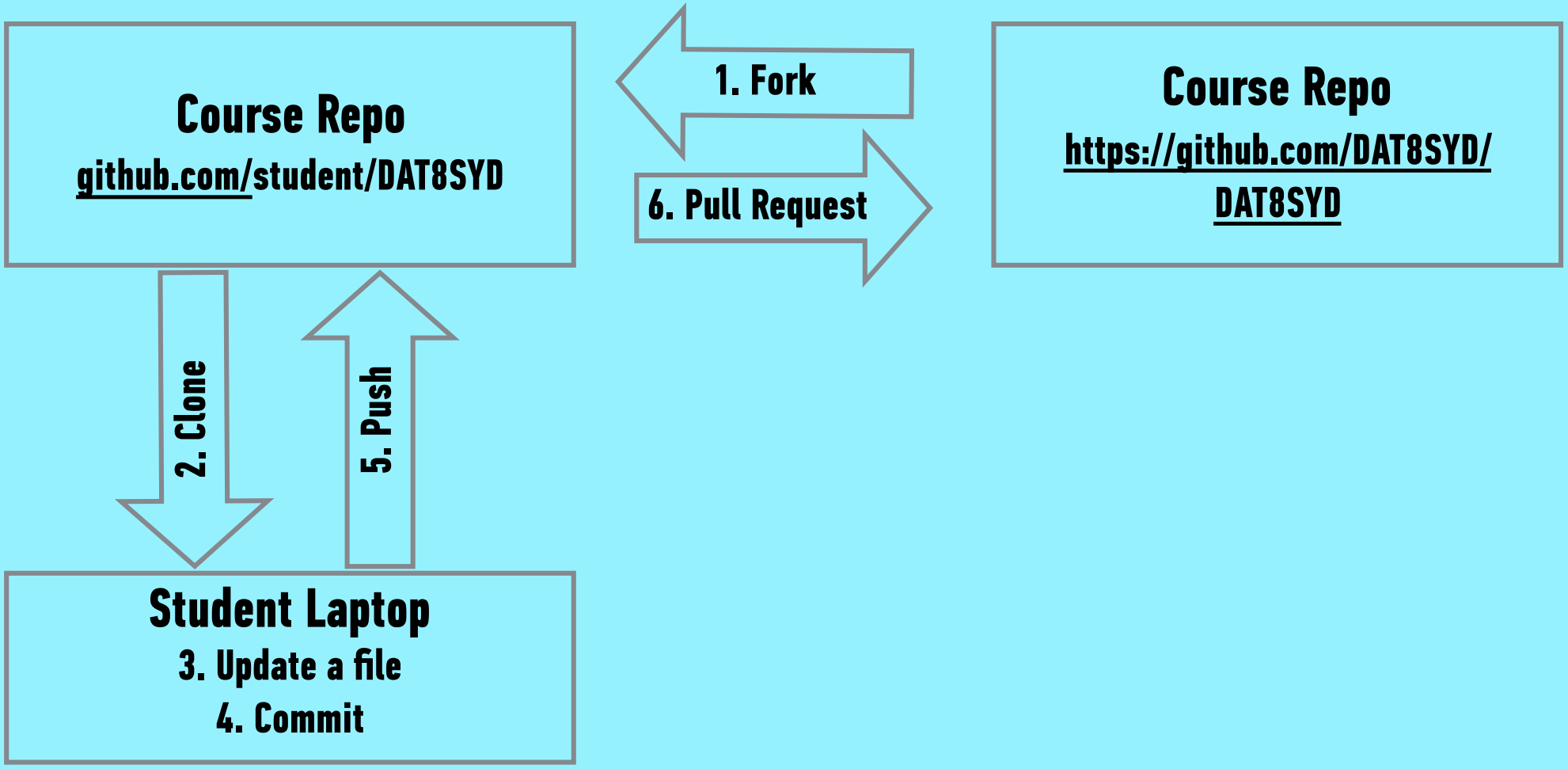
https://help.github.com/articles/good-resources-for-learning-git-and-github/

- A list of common git commands is at the end of this slide deck

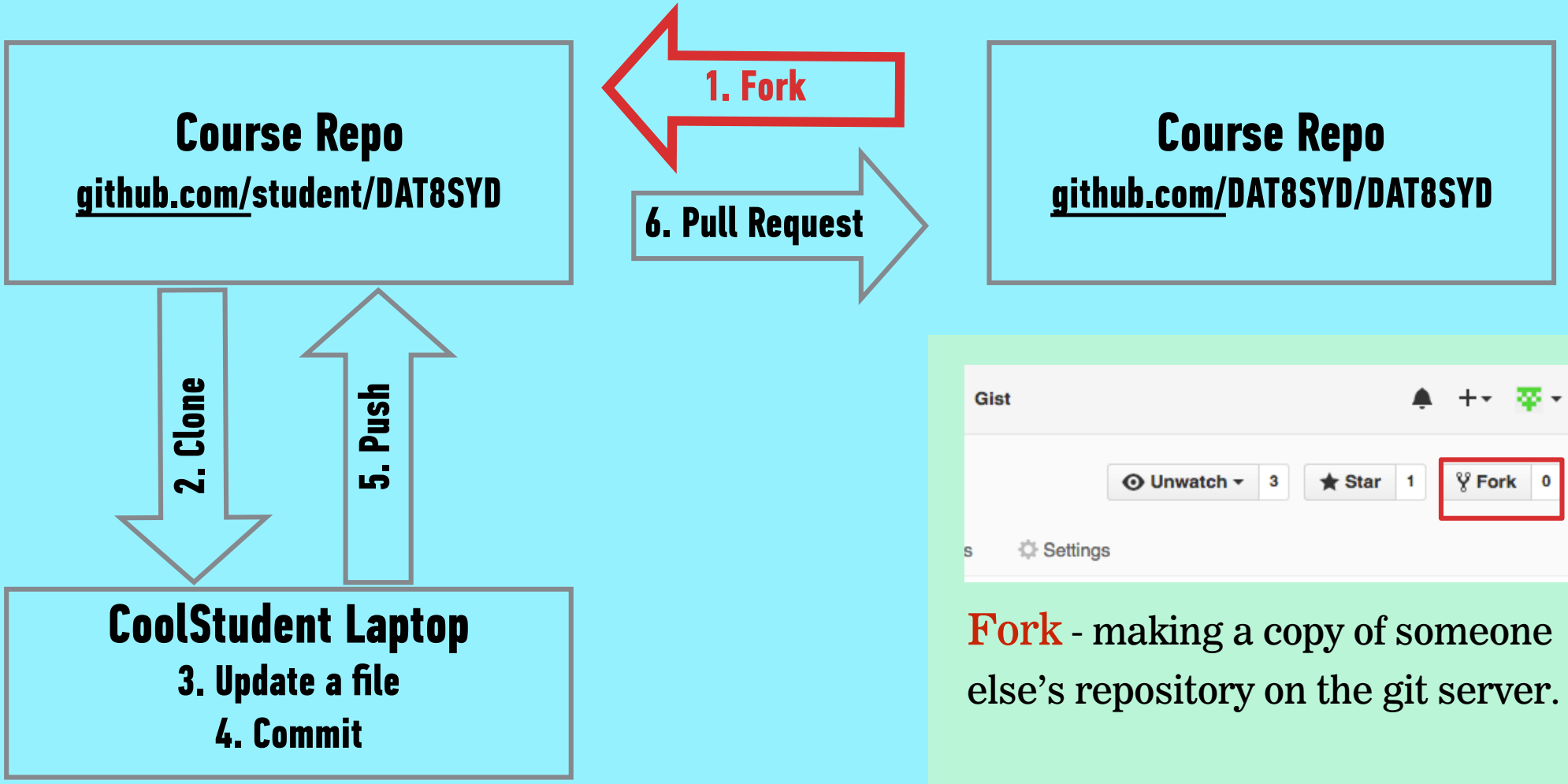
DATA SCIENCE PART TIME COURSE

USING GIT

HOW DO WE ACTUALLY COLLABORATE WITH GIT?

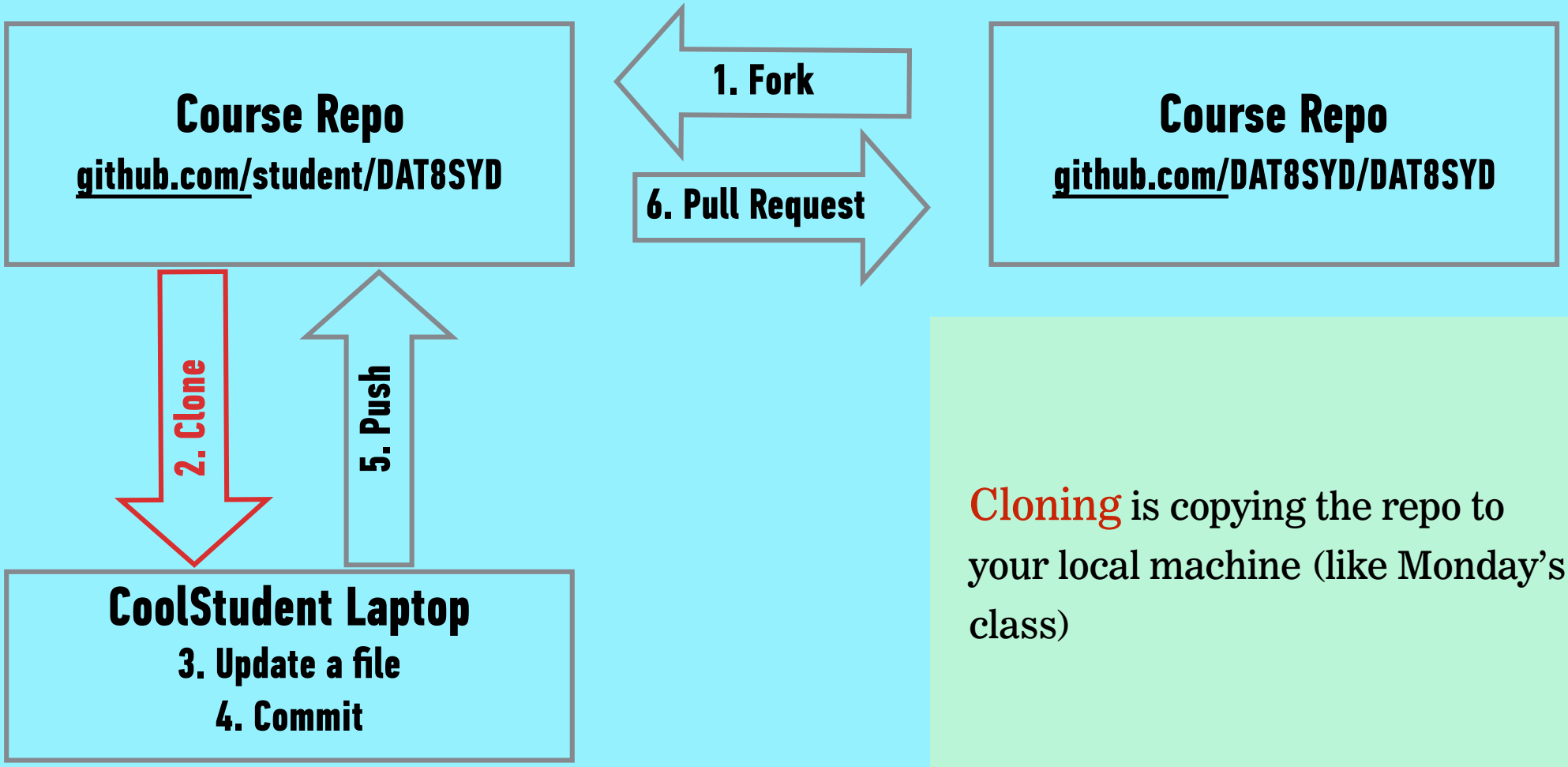


HOW DO WE ACTUALLY COLLABORATE WITH GIT?



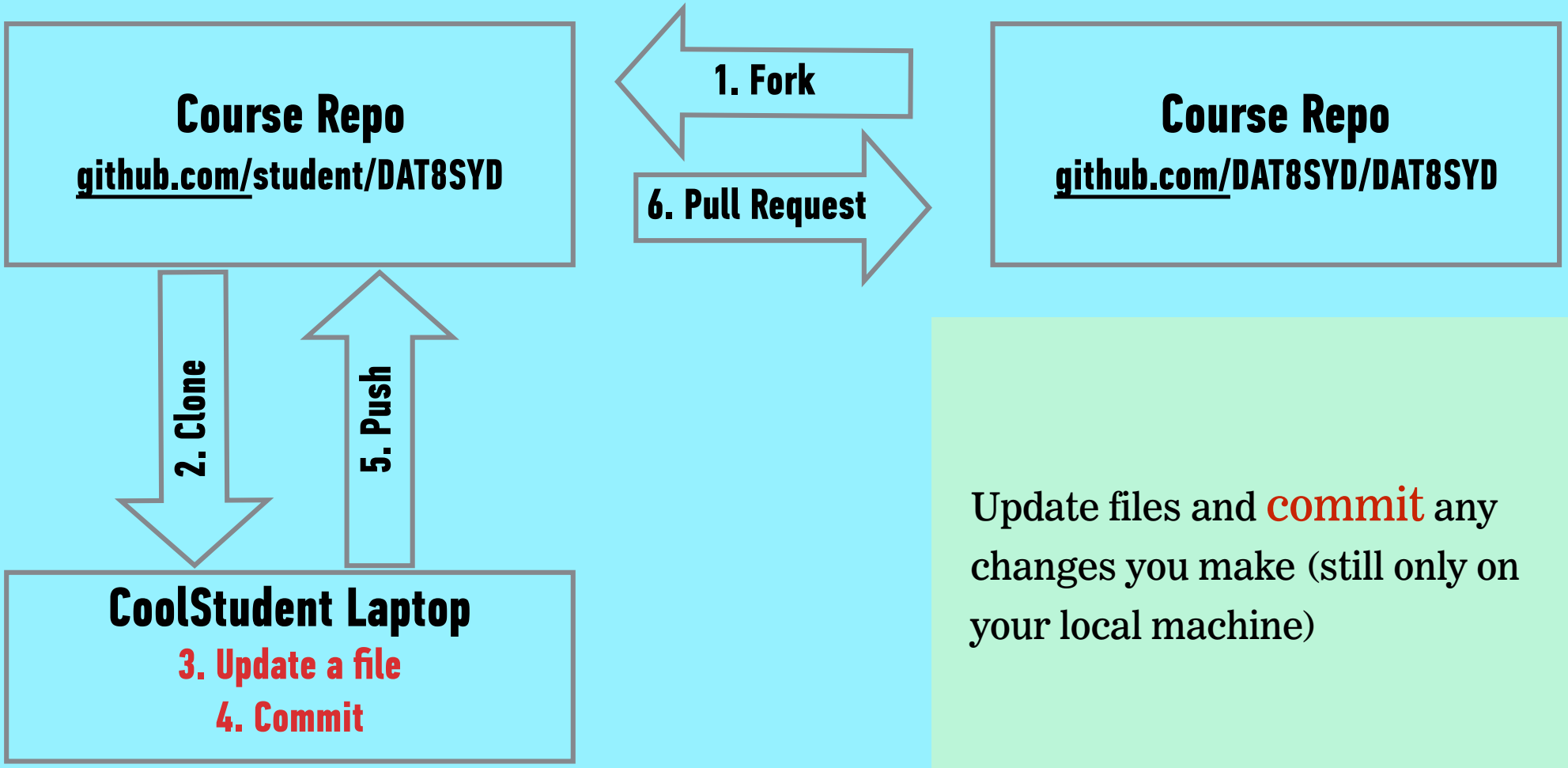
Fork - making a copy of someone else's repository on the git server.

HOW DO WE ACTUALLY COLLABORATE WITH GIT?



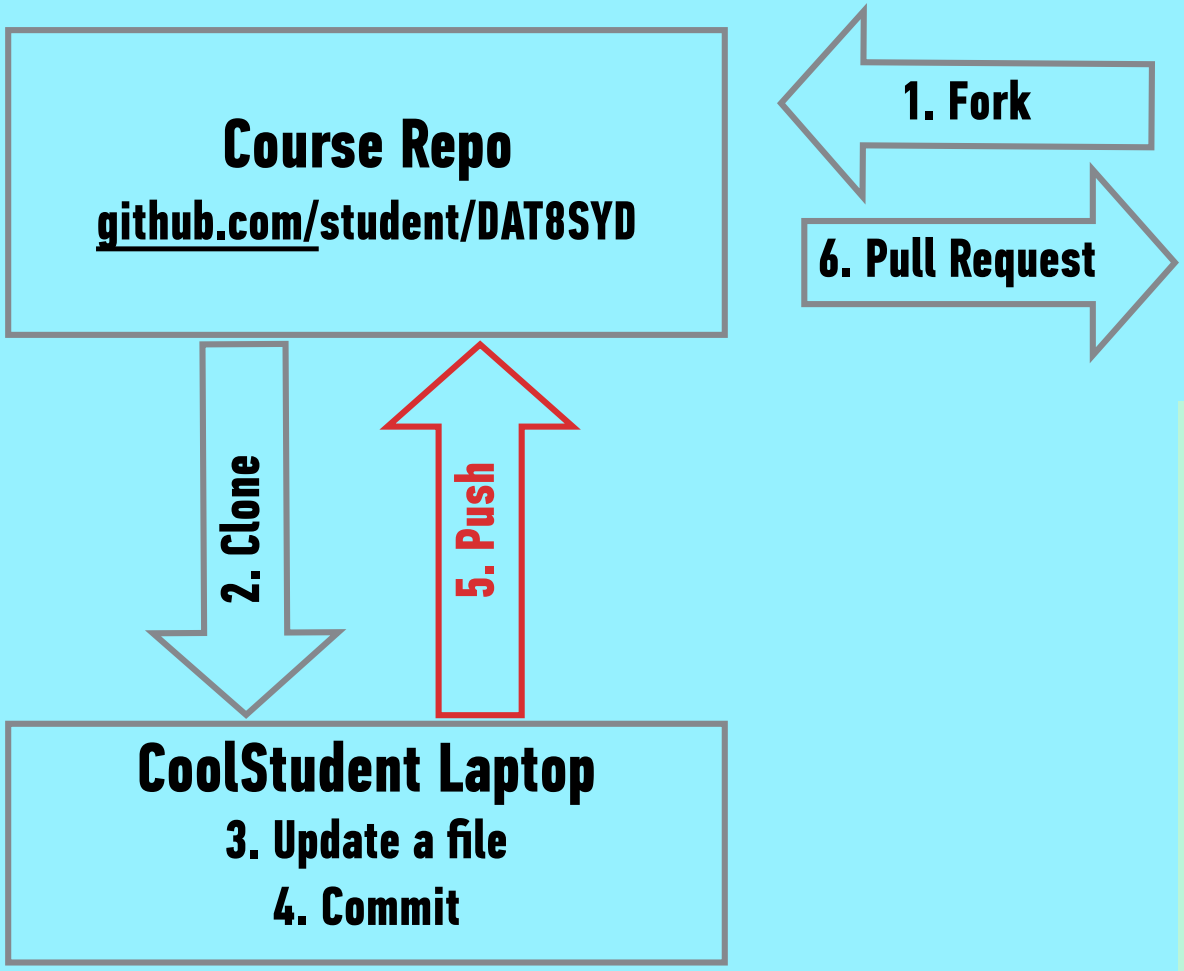
Cloning is copying the repo to your local machine (like Monday's class)

HOW DO WE ACTUALLY COLLABORATE WITH GIT?



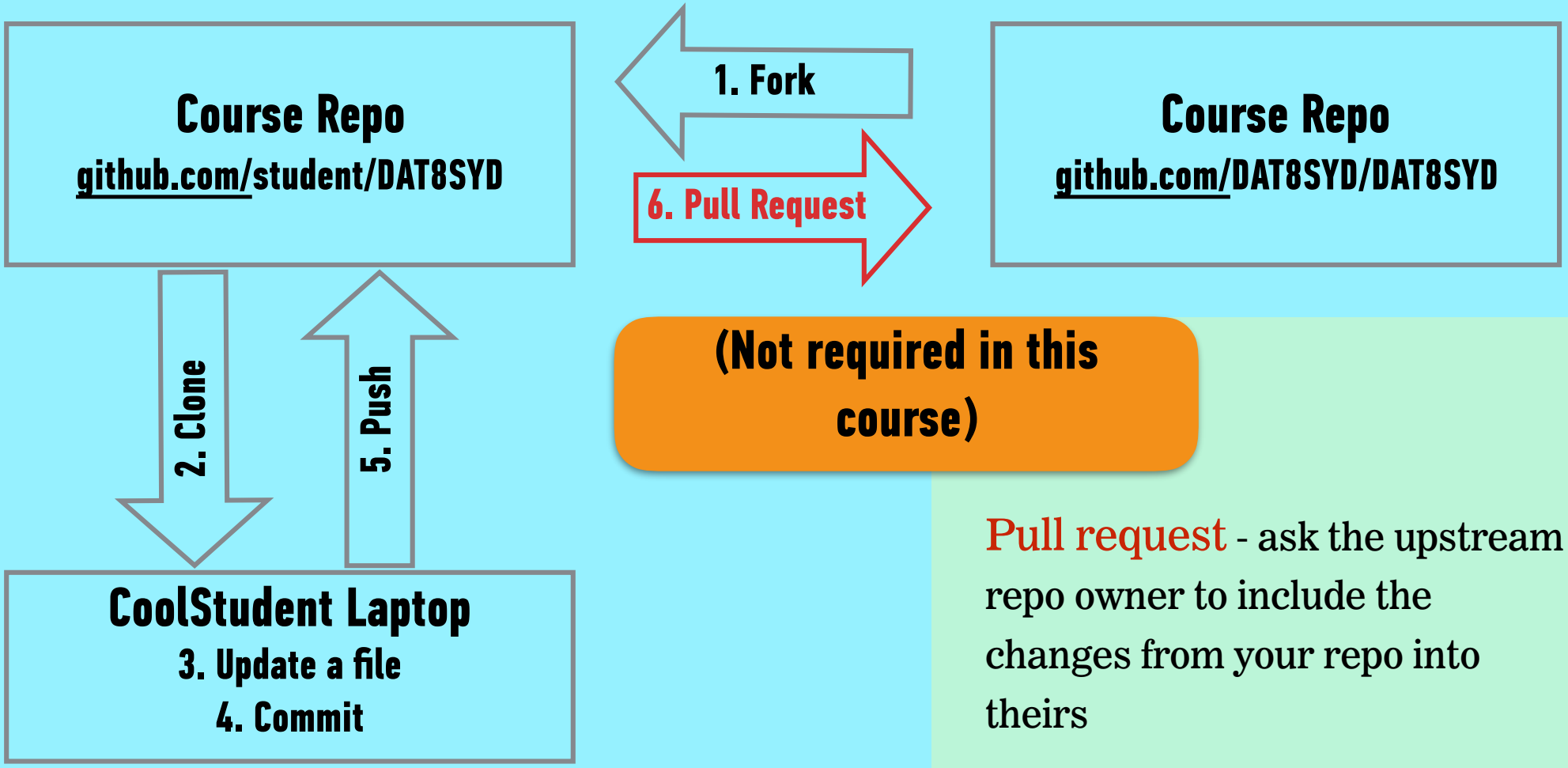
Update files and commit any changes you make (still only on your local machine)

HOW DO WE ACTUALLY COLLABORATE WITH GIT?



Push the changes from your local machine to your github account

HOW DO WE ACTUALLY COLLABORATE WITH GIT?



(Not required in this course)

Pull request - ask the upstream repo owner to include the changes from your repo into theirs

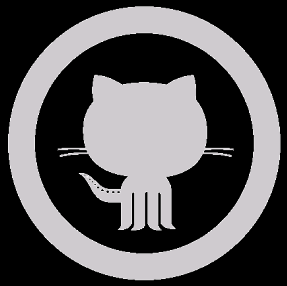
DATA SCIENCE PART TIME COURSE

GIT INSTALL

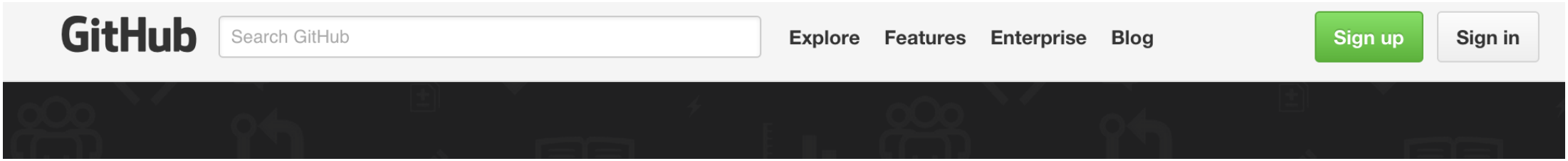
- Installation: goo.gl/MJXSXp
  - Mac: after installation, use Terminal
  - Windows: follow instructions and use git Bash
    - OR, if running Windows 10, try the new "Linux Subsystem on Windows" https://msdn.microsoft.com/en-au/commandline/wsl/faq (Allows you to run Ubuntu Linux natively via Bash.exe)

DATA SCIENCE PART TIME COURSE

INITIAL SETUP



SIGNUP



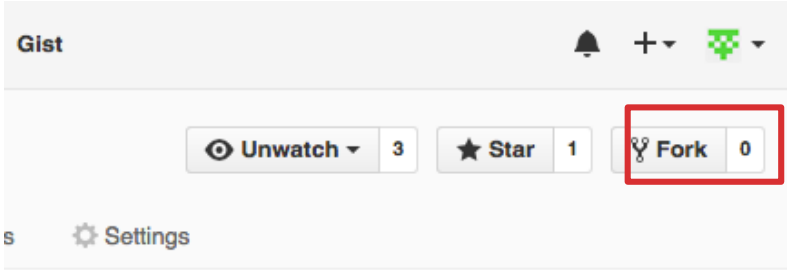
- Click on the signup button on the top-right
- Choose a plan (one of them is free)
- Remember your email and password!!!!

FORK THE COURSE REPO

COURSE REPO: https://github.com/DAT8SYD/DAT8SYD

Fork - making a copy of someone else's repository

- This makes copy of the course repo to your own account. This is your own repo now, but it keeps the history of prior changes and is aware of the source repo you sourced it from



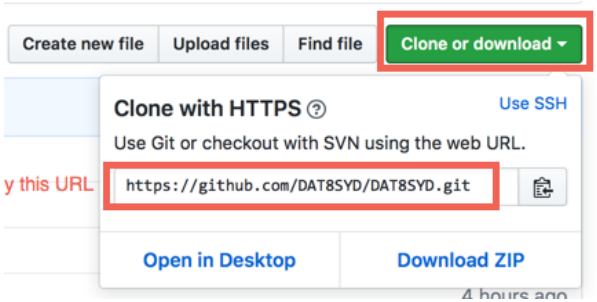
PREVIEW OF WHAT WE ARE ABOUT TO DO

- Copy your new GitHub repo to your computer - **`clone`**
- Make some file changes locally
- Save those changes locally - **`commit`**
- Update your GitHub repo with those changes - **`push`**

- Cloning == copying to your local computer
- Like copying your Dropbox files to a new machine
- First, change your working directory to where you want the repo you created to be stored: cd
- Then, clone the repo: **`git clone <URL>`**
- Get HTTPS or SSH URL from your GitHub (ends in .git)
- Clones to a subdirectory of the working directory
- No visual feedback when you type your password
- Navigate to the repo (cd) then list the files (ls)

CLONE YOUR REPO TO YOUR LOCAL MACHINE

Cloning is copying the repo to your local machine



1. Open a terminal / git bash
2. change your working directory to where you want the repo you created to be stored:

   >cd path-to-where-you-want-to-store-repository

3. clone your copy of the repository from your GitHub account down to your local machine

   > git clone https://github.com/yourgithubname/DAT8SYD.git

4. enter your GitHub username / email and password if prompted.

- Configure user credentials:

```
git config --global user.name “YOUR FULL NAME”
git config --global user.email “YOUR EMAIL”
```

- Use the same email address you used with your GitHub account
- Generate SSH keys (optional): goo.gl/xtH0jJ
- More secure that HTTPS
- Only necessary if HTTPS doesn’t work for you

CONFIGURE THE UPSTREAM SOURCE REPO TO GET UPDATES FROM 24

Configure your local clone (copy) of the repository to point to the official course repository to source regular updates from.

> git remote -v

> git remote add upstream https://github.com/DAT8SYD/DAT8SYD.git

> git remote -v (to check)

origin = your GitHub repo (forked form course repo)

upstream = the course repo

DATA SCIENCE PART TIME COURSE

REGULAR PROCESS

- Check your status:

```
>git status
```

- File statuses (possibly color-coded):
  - Untracked (red)
  - Tracked and modified (red)
  - Staged for committing (green)
  - Committed

- Stage changes for committing:
  - Add a single file: **`git add <filename>`**
  - Add all “red” files: **`git add .`**
- Check your status:
- Red files have turned green
- Commit changes:

```
git commit -m “message about commit”
```

- Check your status again!
- Check the log: git log

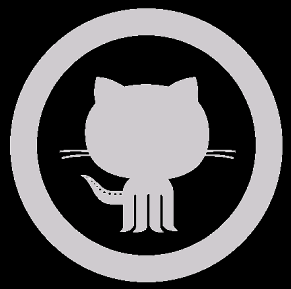
SYNCHING YOUR FORK WITH THE COURSE REPO

1. re-name your labs with lab_name.<yourname>.ipynb (to prevent a conflict)
2. cd <path to the root of your SYD_DAT_6 local repo>
3. commit your changes ahead of sync
   - git status
   - git add .
   - git commit -m “descriptive label for the commit”
   - git status
4. download new material from official course repo (upstream) and merge it
   - git checkout master (ensures you are in the master branch)
   - git fetch upstream
   - git merge upstream/master

- Created a repo on GitHub
- Cloned repo to your local computer - **`git clone`**
- Automatically sets up your "origin" remote
- Made two file changes
- Staged changes for committing - **`git add`**
- Committed changes - **`git commit`**
- Pushed changes to GitHub - **`git push`**
- Inspected along the way - **`git remote`, `git status`, `git log`**

DATA SCIENCE PART TIME COURSE

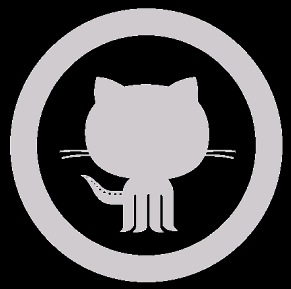
USING GITHUB



- Easy-to-read, easy-to-write markup language
- Valid HTML can also be used within Markdown
- Many implementations (aka “flavors”)
- Let’s edit README.md using GitHub!
- Common syntax:
- ## Header size 2
- *italics* and **bold**
- [link to GitHub](https://github.com)
- * bullet
- `inline code` and ```code blocks```

DATA SCIENCE PART TIME COURSE

COMMON GIT COMMANDS



Basic Git commands

Here is a list of some basic Git commands to get you going with Git.

For more detail, check out the **Atlassian Git Tutorials** for a visual introduction to Git commands and workflows, including examples.

Git task	Notes	Git commands

Tell Git who you are	Configure the author name and email address to be used with your commits.
	
	Note that Git strips some characters (for example trailing periods) from `user.name`.
		`git config --global user.name "Sam Smith"`
		
		`git config --global user.email sam@example.com`

Create a new local repository		`git init`
Check out a repository	Create a working copy of a local repository:	`git clone /path/to/repository`
	For a remote server, use:	`git clone username@host:/path/to/repository`

Add files	Add one or more files to staging (index):	`git add <filename>`
		
		`git add *`

Commit	Commit changes to head (but not yet to the remote repository):	`git commit -m "Commit message"`
	Commit any files you've added with `git add`, and also commit any files you've changed since then:	`git commit -a`
Push	Send changes to the master branch of your remote repository:	`git push origin master`
Status	List the files you've changed and those you still need to add or commit:	`git status`
Connect to a remote repository	If you haven't connected your local repository to a remote server, add the server to be able to push to it:	`git remote add origin <server>`
	List all currently configured remote repositories:	`git remote -v`

Branches	Create a new branch and switch to it:	`git checkout -b <branchname>`
	Switch from one branch to another:	`git checkout <branchname>`
	List all the branches in your repo, and also tell you what branch you're currently in:	`git branch`
	Delete the feature branch:	`git branch -d <branchname>`
	Push the branch to your remote repository, so others can use it:	`git push origin <branchname>`
	Push all branches to your remote repository:	`git push --all origin`
	Delete a branch on your remote repository:	`git push origin :<branchname>`
Update from the remote repository	Fetch and merge changes on the remote server to your working directory:	`git pull`
	To merge a different branch into your active branch:	`git merge <branchname>`

	View all the merge conflicts:
	View the conflicts against the base file:
	Preview changes, before merging:
		`git diff`
		`git diff --base <filename>`
		`git diff <sourcebranch> <targetbranch>`

	After you have manually resolved any conflicts, you mark the changed file:	`git add <filename>`
Tags	You can use tagging to mark a significant changeset, such as a release:	`git tag 1.0.0 <commitID>`
	CommitId is the leading characters of the changeset ID, up to 10, but must be unique. Get the ID using:	`git log`
	Push all tags to remote repository:	`git push --tags origin`
Undo local changes	If you mess up, you can replace the changes in your working tree with the last content in head: Changes already added to the index, as well as new files, will be kept.	`git checkout -- <filename>`
	Instead, to drop all your local changes and commits, fetch	`git fetch origin`

Undo local changes	If you mess up, you can replace the changes in your working tree with the last content in head: Changes already added to the index, as well as new files, will be kept.	`git checkout -- <filename>`

	Instead, to drop all your local changes and commits, fetch the latest history from the server and point your local master branch at it, do this:	`git fetch origin`
		
		`git reset --hard origin/master`

Search	Search the working directory for `foo()`:	`git grep "foo()"`