

...and, you made it!

What'd you just do?

Remember how we learned that git is *distributed*? Your personal machine is your 'local'. Other machines that you talk to with git are 'remotes'. GitHub is a remote. You just told your computer to go to a remote and pull down a repository (aka, a directory, or folder).

See it for yourself

```
~/Documents ⚡ cd rocketu-201401/  
~/Documents/rocketu-201401 master ⚡ git remote  
origin  
~/Documents/rocketu-201401 master ⚡
```

The command 'git remote' will list all the remotes you have associated with a repository. Note that GitHub will always call itself 'origin' by default. Later today, you'll see what it's like when we add Heroku as another remote.

Basic git workflow

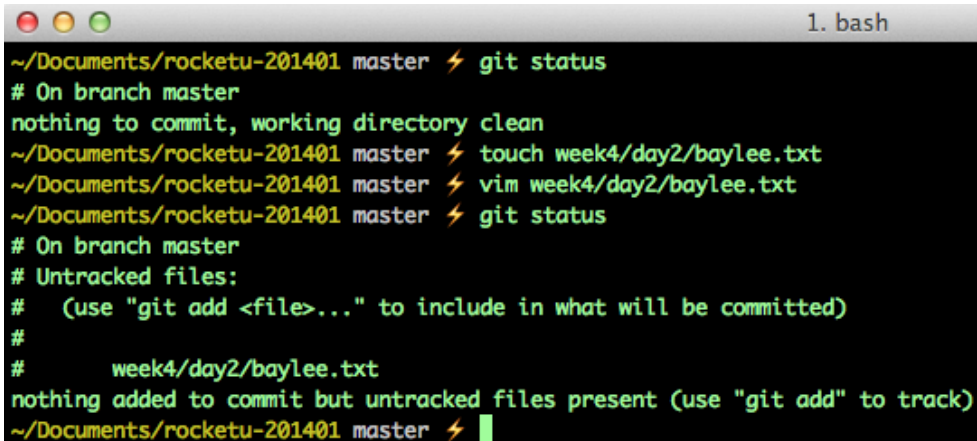
Init or clone, change, add, commit

Checking status and making changes

Making changes

Git isn't going to do anything until you tell it to. This is not Google Drive autosave. When you add a new file or make changes to an existing file tracked by git, it is aware of those changes, but doesn't do anything except acknowledge the changes.

What does this look like?

A terminal window titled "1. bash" with a dark background and light green text. It shows a sequence of commands and their outputs. The user is in the directory ~/Documents/rocketu-201401 on the master branch. They run 'git status', which reports a clean working directory. Then they run 'touch week4/day2/baylee.txt' and 'vim week4/day2/baylee.txt'. Finally, they run 'git status' again, which reports that there are untracked files (week4/day2/baylee.txt) and suggests using 'git add' to track them.

```
~/Documents/rocketu-201401 master ✨ git status
# On branch master
nothing to commit, working directory clean
~/Documents/rocketu-201401 master ✨ touch week4/day2/baylee.txt
~/Documents/rocketu-201401 master ✨ vim week4/day2/baylee.txt
~/Documents/rocketu-201401 master ✨ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       week4/day2/baylee.txt
nothing added to commit but untracked files present (use "git add" to track)
~/Documents/rocketu-201401 master ✨
```

Tracking / staging changes

Tracking Changes

Since we want to track these files in our version control, we will add them to git. You do this with the command `git add`

```
~/Documents/rocketu-201401 master ✨ git add week4/day2/baylee.txt
~/Documents/rocketu-201401 master ✨ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   week4/day2/baylee.txt
#
~/Documents/rocketu-201401 master ✨ git reset
~/Documents/rocketu-201401 master ✨ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       week4/day2/baylee.txt
nothing added to commit but untracked files present (use "git add" to track)
~/Documents/rocketu-201401 master ✨ git add .
~/Documents/rocketu-201401 master ✨ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   week4/day2/baylee.txt
#
~/Documents/rocketu-201401 master ✨
```

Add this file explicitly. When you do 'git status' again, you'll see that the file is now tracked

'git reset' unstages your changes (it takes you back to the last commit). Now the file is untracked again.

Use 'git add .' to add all changes to the staging area

Commit = Save

Once you're happy with your changes, want to stop, or are at a good break point, you're ready to commit. Committing is the same as saving.

```
~/Documents/rocketu-201401 master ⚡ git commit -m "Introduction to baylee"
[master c2957ee] Introduction to baylee
1 file changed, 1 insertion(+)
create mode 100644 week4/day2/baylee.txt
~/Documents/rocketu-201401 master ⚡
```

Understanding revision history and the git tree

History

(each circle is a commit, or save point)



In any single commit:
Make some changes, add
them, commit them

Make more changes, add
them, commit them

See the history of your
commits by running 'git log'

Your git history is a tree. Move around the tree with 'git checkout'. E.g., 'git checkout 2' will take you to commit #2

Do this: change, add, commit

- Make a `firstname_lastname.txt` file
- Add it
- Commit it
- Try using the following commands before / after each step:
 - `git status`
 - `git diff`
 - `git log`

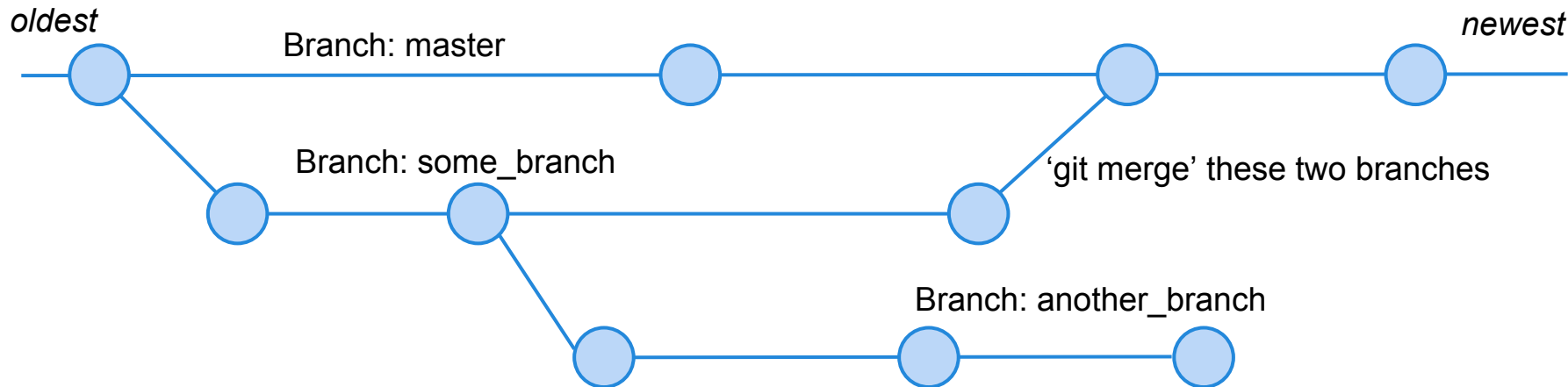
Intermediate git workflow

Branching and merging

Git history: it really is a tree

History

(each circle is a commit, or save point)



Each line is a branch. Remember when I said you could move around your git tree with 'git checkout'? Not only does it cover moving up and down commits in one branch, it also allows you to move to other branches. Combine changes from two branches using 'git merge'.

Do this: practice branching

- cd out of my repository and into your project folder
- Initialize it as a git repo if you haven't yet ('git init')
- Add and commit any changes
- Checkout a new branch (git checkout -b newbranch)
- List all your branches (git branch)
- Try something new you've been scared of doing because you've been afraid of losing your work
- Add and commit your changes
- Check your git log
- Move back to your master branch (git checkout master)
- Check your git log
- Try sketching your current git revision tree

Do this: practice merging

Simple Merge

- Checkout another new branch (`git checkout -b another_branch`)
- Add and commit some changes
- Move back to your master branch
- Merge your branch into your master branch (`git merge`)
- Use `git log` to see your changes
- Try to sketch out your git revision tree

Ruh-roh. Merge conflicts.

You should notice from the times you've run 'git diff' that git tracks changes *by line*. Of course, it's not uncommon for changes to be made on one branch, and other changes to that same line to be made on another branch. When you try to merge these two branches, git doesn't know what to do.

Instead, git will put you in no man's land until you fix the merge conflict. You do this by picking which changes you want to keep or discard. Then, you add and commit the changes, and you're back on track.

Let's see how that looks.

Do this: Fix a merge conflict

Simple Merge

- Checkout another new branch
- Add and commit some changes
- Move back to your master branch
- Change a file ***on the same line*** that you changed a file in your other branch
- Merge your branch into your master branch
- Use the merge tool of your choice (vim, Xcode developer tools, Sublime, etc.) to fix your conflict
- Commit your revisions
- Complete merge
- Use git log to see your changes
- Try to sketch out your git revision tree

Playing nice with others

Pull, pull requests, and finding a workflow

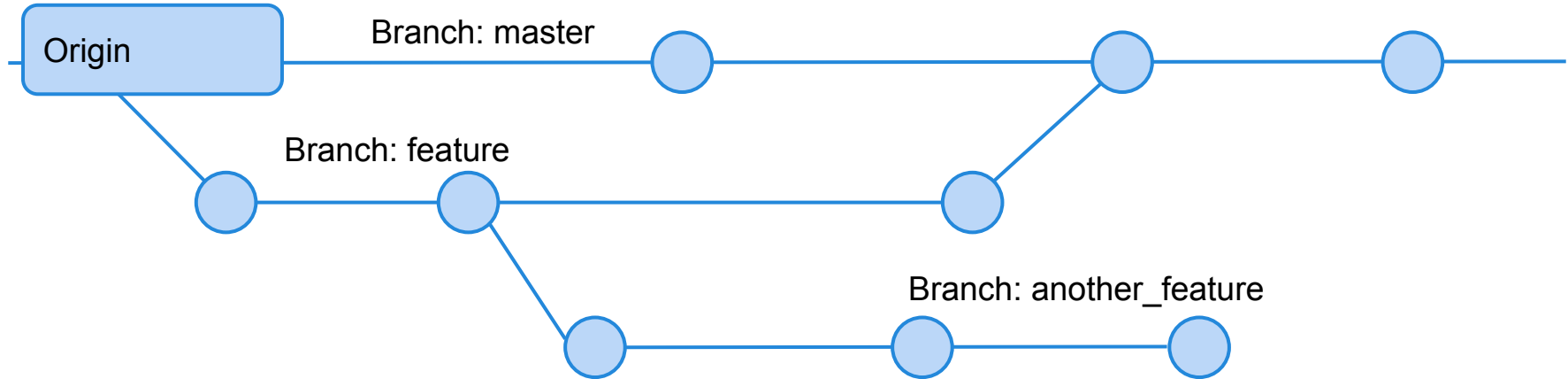
Git pull, and thinking about remotes

So far everything you've done has been on your local machine. Obviously, one of the purposes of git is to allow collaboration. You will have a remote (like Github) as your source of truth, but because git is distributed, your machine can have a branch structure different from 'origin', and anyone else collaborating might also have a different structure.

You use commands like 'git push origin feature_branch' or 'git pull origin master' to refer to specific branches. 'git pull' will merge the latest changes from a remote branch.

Feature branches

A common git workflow for a team might look like this:



Every single time you work on a new feature, you create a new branch. Then you push that branch to origin, creating a new branch at the remote. You use 'pull requests' to merge changes into master.

Do this: Commit and push a feature branch

- Make a GitHub account if you don't have one yet. Make sure your GitHub account is linked to your computer. This should be explained in the account setup process, but let me know if you are having trouble.
- Remember the `firstname_lastname.txt` file you made earlier in your clone of my repo? Add that file to a new branch
- Push that *branch* to origin (my repo on GitHub) -- do NOT push to master
- Submit a pull request to merge the branch into master

The rest of the
iceberg

Git has many other cool features

If you want to learn more...

- Read the rest of the git book assigned as homework last night
- Git is one of the best documented web technologies, with lots of beginner material. There are online, interactive git tutorials, lots of great YouTube videos, and plenty of cheatsheets.
- A few things to check out:
 - <http://pcottle.github.io/learnGitBranching/>
 - <http://byte.kde.org/~zrusin/git/git-cheat-sheet-large.png>
 - <http://nvie.com/posts/a-successful-git-branching-model/>