Git Workshop, Desjardins Lab, Montreal, QC June 11, 2016

An introduction to collaborative coding and open source via the wonderful world of version control.
Notes by Paige Sun. I would be happy to hear your thoughts at paige.sun@cs.mcgill.ca!
Instructed by Arielle Vaniderstine, Florent Vilmar, and Étienne Caron. *Hosted by Les Pitonneux.*

What is Git?

- **Git** is a tree of history of the all the changes in the code.
- A **commit** is a snapshot of all the code, in a frozen point in time in the history of the entire project. It includes the changes in the code that happened to get from the previous commit to this commit.

For Terminal and Vim commands, please see the last page.

View Current Status

git status
- Status of the branch you're on. Generally tells you what to do next.

git branch -a
- See all the branches for this repository, local and remote.

git diff <filename>
- **q** to exit diff mode.

View History

git log
- View history of commits with SHA, Author, Date, and Message.

git log --pretty=oneline
- View history of commits with only SHA and message.

git lol
- View history of commits as ascii art branches.
- One-time setup: **git config --global alias.lol "log --pretty=oneline --abbrev-commit --graph --decorate"**

git reflog
- View history of *all* git commands.

Create, Add, and Commit

git init
- Create a new repository.

git add <filename1> <filename2>
- Stage filename1 and filename2 so that the changes may be committed.

git add -A
- Stage *all* files in the directory.

git commit -m “First commit”
- A **commit** is the making of a set of tentative changes permanent. Write commit messages in present tense.

Branching

git branch
- See all the local branches in the repository.

git branch -a
- See all the local and remote branches in the repository.

git checkout -b <branch_name>
- Make a new branch and switch onto that branch.
- Basically, it's two commands combined into one:
  - **git branch <branch_name>** #create a new branch, but stay on current branch
  - **git checkout <branch_name>** #switch to an existing branch

Option 1: Connecting and existing remote repo to an existing local repo

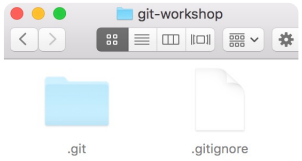
git remote add origin https://github.com/CitronDigital/git-workshop

git remote -v
- To see a list of a Git project's remotes)

Option 2: Connecting to an remote repo when you don't have an existing local repo

git clone https://github.com/CitronDigital/git-workshop

cd git-workshop # To navigate into your local repository folder. --->

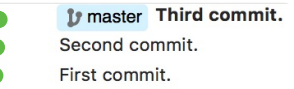


Lab 1 - Fetching from remote, and merging your changes on top.

- Let's make some commits! Here's the info displayed with **git lol** and **SourceTree**.
  - Here, eac circle ● represents a commit, coloured by which branch that the commit came from.

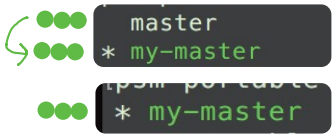
```
$ git branch -a
* master
```

```
* 3b26319 (HEAD -> master) Third commit.
* e894121 Second commit.
* b5d021c First commit.
```



- **git remote add** origin https://github.com/CitronDigital/git-workshop
  - git remote add <what_you_want_to_call_the_server> <url_of_the_server>
  - Assign the name origin to refer to the location of the remote server hosting a git repository.

```
[psm-portable-03:gitexp psun$ git remote -v
origin  https://github.com/CitronDigital/git-workshop (fetch)
origin  https://github.com/CitronDigital/git-workshop (push)
```



- **git checkout -b my-master**
- **git branch -D master**
  - Delete the old 'master' that was automatically added when we did **git init.**

- **git fetch**
  - List and get all new changes in this repository from the remote, so that we have locally on our machine, the state of the remote repository since the beginning of time.
  - Note: You generally want to do **git fetch**, and not **git pull**
    - b/c git pull will trigger a merge, and you might not want to merge without looking at the new code first.

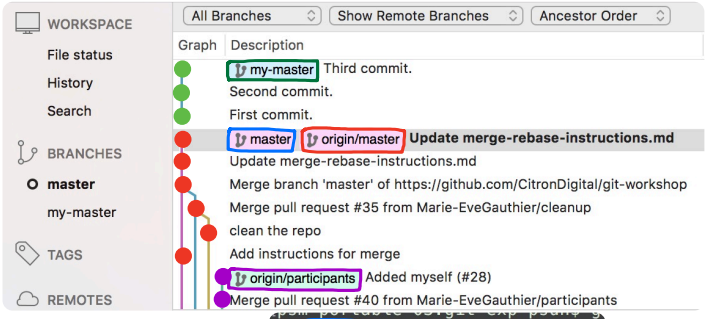
```
* my-master
  remotes/origin/master
  remotes/origin/participants
```

- **git checkout origin/master**
  - Now you are in 'detached HEAD' state.
    - If you're on a detached head state, your commits will be discarded when you switch to another branch. You must make a new branch first to save commits.

```
* (HEAD detached at origin/master)
  my-master
  remotes/origin/master
  remotes/origin/participants
```

- **git checkout -b master**
  - Make a new local **master** branch that has all the new files from **origin/master**
  - Note: We deleted the **initial master branch** so that we can create this **new master** branch.

```
* master
  my-master
  remotes/origin/master
  remotes/origin/participants
```



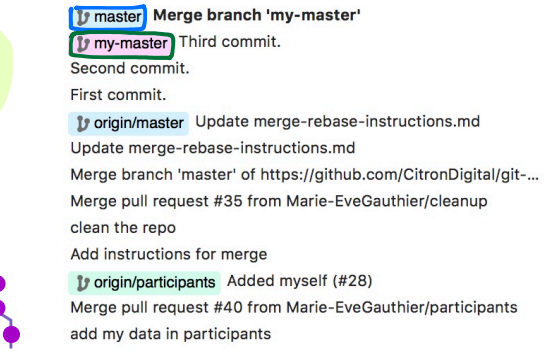
Now we have the branches:

- **local my-master** with all the with the commits you've made before you fetched.
- **local master** which matches the contents of the **remote origin/master**
- Another **remote origin/participants**

- **git merge my-master**
  - merge **local my-master** onto your new **local master**
  - Type **escape** : **x** to save and quit vim

```
*   5a5d21b (HEAD -> master) Merge branch 'my-master'
|\
| * 3b26319 (my-master) Third commit.
| * e894121 Second commit.
| * b5d021c First commit.
* 414a012 (origin/master) Update merge-rebase-instructions.md
* f5d0591 Update merge-rebase-instructions.md
* 55ca0f9 Merge branch 'master' of https://github.com/CitronDigital/g
|\
| * 182e722 Merge pull request #35 from Marie-EveGauthier/cleanup
| |\
| | * 408046e clean the repo
| |/
| * d4a9dc4 Merge pull request #18 from cbredeau/master
| |\
```

```
* master
  my-master
  remotes/origin/master
  remotes/origin/participants
```



Theory:
Local branch my-master: 1-2-3
Local branch master: A-B-C <---- **local master** matches the contents of the **remote origin/master**

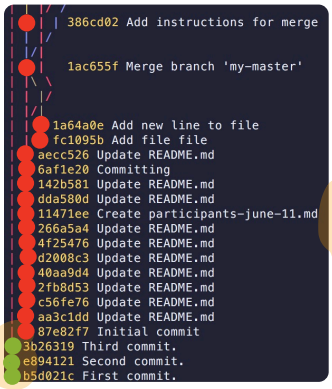
Ultimately, we want **local master** to have the commits A-B-C-1-2-3, not 1-2-3-A-B-C, so that we can push **local master**'s merged contents into **remote origin/master**

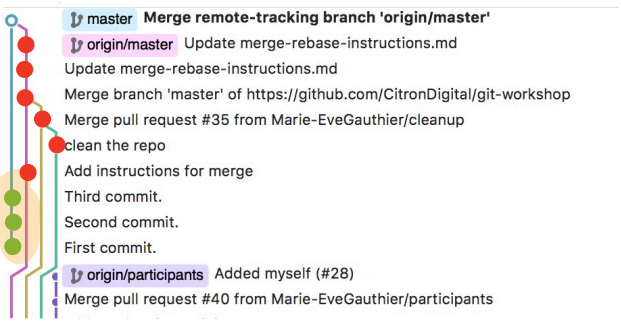
- Why? The branch master has the stable product. To keep the changes clean and sequential, and make it easier to modify our code after we commit, we want to put our code after the good code already in the master.

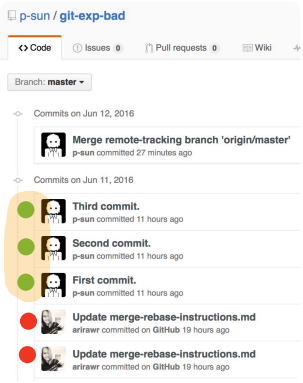
Above is a ***good*** merge practice, below is a ***bad*** merge practice.

- Make three local commits
- **git remote add citrondigital** https://github.com/CitronDigital/git-workshop
- **git fetch**

- **git merge citrondigital/master**







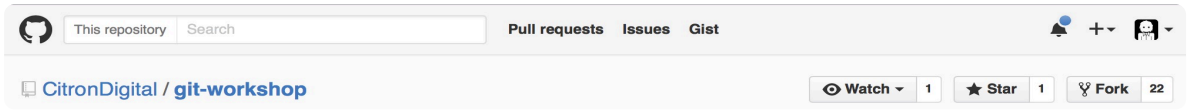
$git lol SourceTree GitHub

Lab 2 - Adding Remote Commits onto Local Branch

- **git remote rename origin citrondigital**
  - Change the server name from **origin** to **citrondigital**

```
psm-portable-03:git-exp psun$ git remote -v
citrondigital   https://github.com/CitronDigital/git-workshop (fetch)
citrondigital   https://github.com/CitronDigital/git-workshop (push)
```

- On GitHub, click the "fork" button to make a copy of CitronDigital repo in your own GitHub account.



  - Note: Since you made a copy of the repo citrondigial/git-workshop into your own repo username/git-workshop, you now also have two branches on your own repo named "master" and "participants" with identical contents.
- **git remote add origin** https://github.com/*username*/git-workshop

```
psm-portable-03:git-exp psun$ git remote -v
citrondigital   https://github.com/CitronDigital/git-workshop (fetch)
citrondigital   https://github.com/CitronDigital/git-workshop (push)
origin  https://github.com/p-sun/git-workshop (fetch)
origin  https://github.com/p-sun/git-workshop (push)
```

- **git fetch origin**

```
* master
  my-master
  remotes/citrondigital/master
  remotes/citrondigital/participants
  remotes/origin/master
  remotes/origin/participants
```

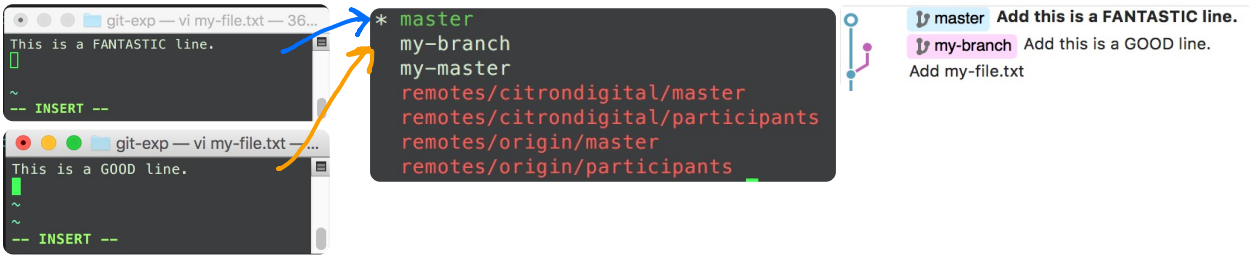
  - Get the full repository from the server origin onto your computer, so you can access the contents on the two remote branches named **origin/master** and **origin/participants.**
    - origin is the server url https://github.com/*username*/git-workshop

Lab 3 - Creating a conflict, resolving the conflict, and pushing new changes onto a remote branch

- In **master** branch:
  - **vi my-file.txt** -> **i** -> insert "This is a BAD line." -> **esc :x** to save and exit vi
  - **git add** my-file.txt; **git commit** -m "Add this is a BAD line";

- Inside **my-branch**:
  - **git checkout -b my-branch**
  - **vi my-file.txt** -> **i** -> edit to "This is a GOOD line." -> **esc :x**
  - **git add** my-file.txt; **git commit** -m "Add this is a GOOD line";

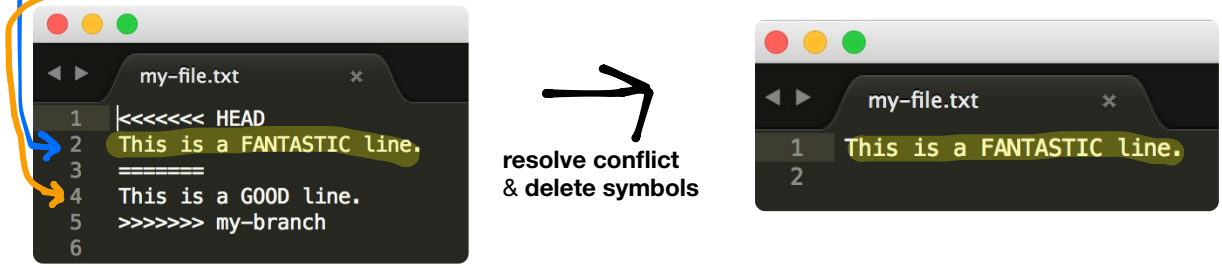
- Go back to **master**:
  - **git checkout master**
  - **vi my-file.txt** -> **i** -> edit to "This is a FANTASTIC line." -> **esc :x**
  - **git add** my-file.txt; **git commit** -m "Add this is a FANTASTIC line";



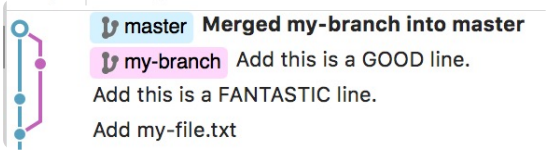
- **git merge my-branch**
  - merge **my-branch** onto the **HEAD** branch, **master**.
  - You've now created a conflict because **master** and **my-branch** differ on the same line!

```
[psm-portable-03:git-exp psun$ git merge my-branch
Auto-merging my-file.txt
CONFLICT (content): Merge conflict in my-file.txt
Automatic merge failed; fix conflicts and then commit the result.
```

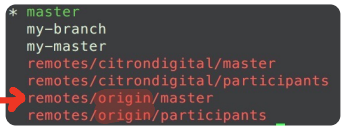
- Open my-file.txt in your favourite text editor. Delete the symbols and the code you don't want to keep.
  - Between <<<<<<< and ======== are from the **HEAD** branch, which is currently **master**.
  - Between ======== and >>>>>>> are from the branch **my-branch.**



- **git add** my-file.txt; **git commit -m** "Merged my-branch into master";
  - After we merged the files,

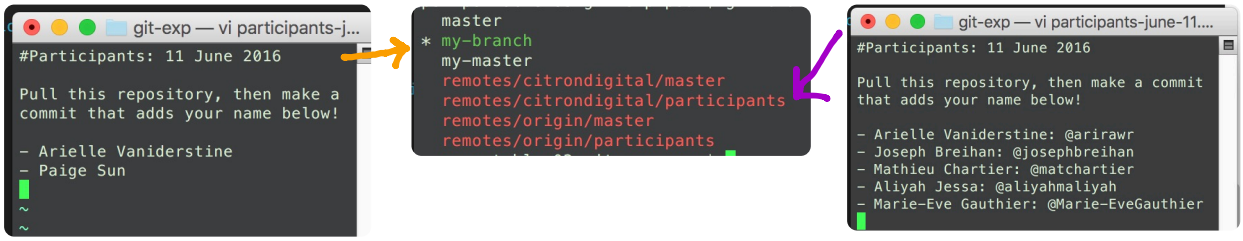


- **git push origin master**
  - Push **current branch** onto the **remote master branch** on the **origin server**
    - **current branch** = local master branch
    - **origin** = https://github.com/p-sun/git-workshop
    - **remote master branch** on the **origin server**



Lab 4A - Editing the participants list, resolving the conflict, and pushing the changes.
Method 1: The quick and dirty way using 'merge'.

- **git checkout** my-branch
- Add your name to the participants list:
  - **vi** participants-june-11.md -> **i** to enter *insert mode* -> type your name -> **esc :x** to save

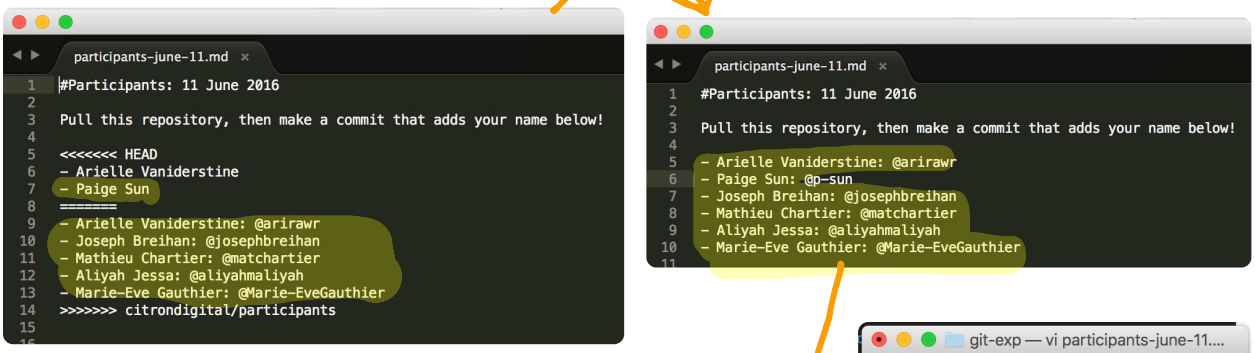


- **git add** participants-june-11.md; **git commit -m** "Add my name on the second line."
- **git fetch citrondigital**
- **git merge** citrondigital/participants
  - Merge the **remote citrondigital/participants** into the current branch, the **local my-branch.**
    - Now you have conflicts! Resolve conflicts as explained in the previous page, save the file.

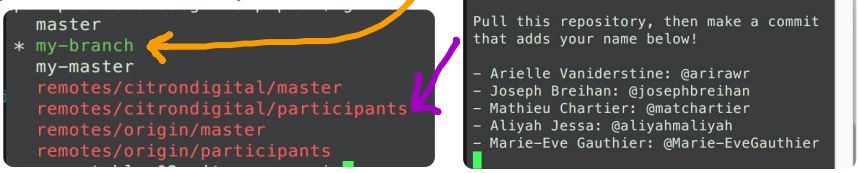
Merge

- **vi** participants-june-11.md
- -> i to enter *insert mode*
- -> type your handle and resolve conflicts
- -> **esc :x** to save

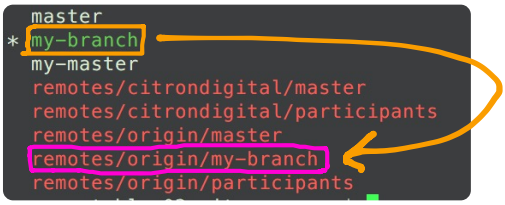
Resolve Conflict



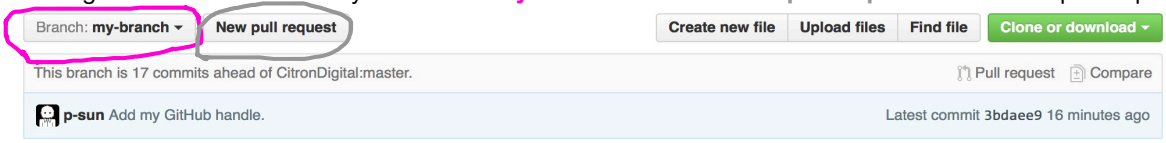
- **git add** participants-june-11.md; **git commit -m** "Add my GitHub handle."



- **git push** origin my-branch
  - Push the current branch, the **local my-branch**, onto the **remote my-branch** on the **origin server**.
  - Since **origin/my-branch** existed before, a new remote my-branch is created on your GitHub account!



- Now go on GitHub -> choose your **remote my-branch** -> **click "New pull request"** -> "Create pull request"



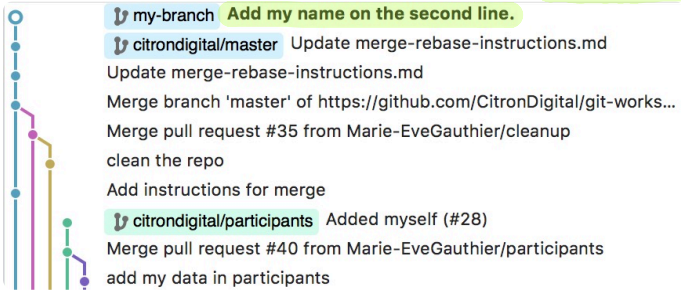
Lab 4B -- Editing the participants list, resolving the conflict, and pushing the changes.
Method 2: Adding your new changes with the power method by using 'rebase', instead of 'merge'.

Instead of **git merge citrondigital/participants** that we did previously,
We can do **git rebase citrondigital/participants**

To make diagrams easier to understand, I removed all the branches that we're not using.

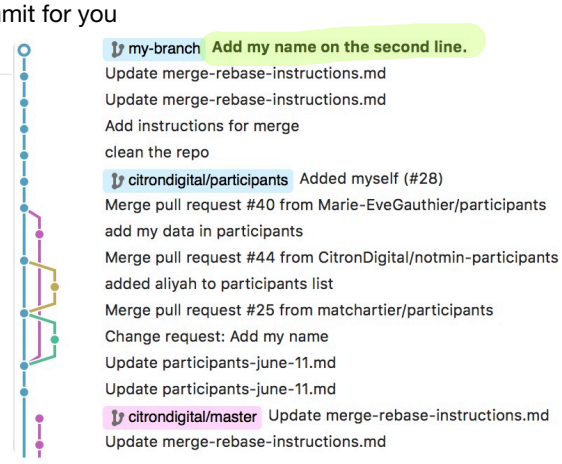
```
* my-branch
  remotes/citrondigital/master
  remotes/citrondigital/participants
```

Step 0 recap: method 1 and method 2 starts off the same:
- git checkout citrondigital/master
- git branch -D my-branch
- git checkout -b my-branch
- vi participants-june-11.md -> i to enter *insert mode* -> type your name -> esc :x to save
- git add participants-june-11.md; git commit -m "Add my name on the second line."



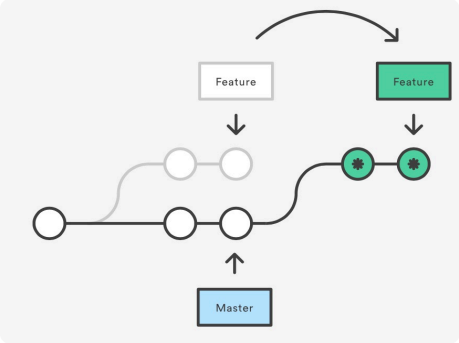
After step 0, use method 1(left):
- **git merge citrondigital/participants**
- Resolve conflicts: vi participants-june-11.md -> i to enter *insert mode* -> resolve conflicts -> esc :x to save
- git add participants-june-11.md; git commit -m "Add my GitHub handle."
- git push -f origin my-branch

After step 0, use method 2 (right diagram):
- **git rebase citrondigital/participants**
  - Rewinding to replay your work on top of **citrondigital/participants**
  - **git rebase --abort**
    - if something goes wrong
- Resolve conflicts: vi participants-june-11.md -> i to enter *insert mode* -> resolve conflicts -> esc :x to save
- git add participants-june-11.md
- **git rebase --continue**
  - Don't **git commit** here, because git rebase will commit for you
- git push -f origin my-branch



- Note how much cleaner **rebasing** (diagram right) is compared to **merging** (diagram left)!
- Also note that **rebasing** only added one commit message, while **merging** required two!

- What is rebasing?
  - The process of moving a branch to a new base commit.
    - Let's say you created a **new feature branch** and you want to add your new features to master.
    - '**git rebase master**' will use the master as the base, and then replay **your new changes** on top of your master.
    - By rebasing instead of merging, we can keep a clean linear commit history, preserve the order of new change-sets, and avoid clogging the history with many commits from merging. i.e. *merge branch 'abc'*
    - Preference of merging vs rebasing depends on the team.



- When do you want to rebase?
  - When your local branch have conflicts with the remote.

- The Git workflow
  - The **master branch** is always production-ready, deployable, and can pass all the tests.
  - New development is done on **feature branches**.
  - You make **commits**, **rebase**, **push** to remote, and then make a **pull request** for review.

Lab 5A - Interactive rebasing: squashing commits

- Create a new git repo. i.e.
  - **cd; cd Documents; mkdir git-rebasing; cd git-rebasing; git init**

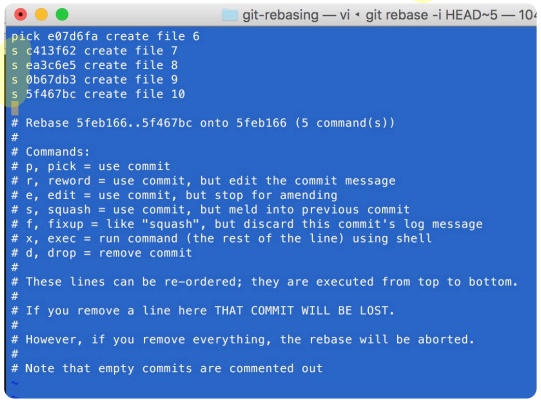
```
psm-portable-03:git-rebasing psun$ git branch -a
* master
```

- Make 10 new files quickly, and make a new commit for every new file.

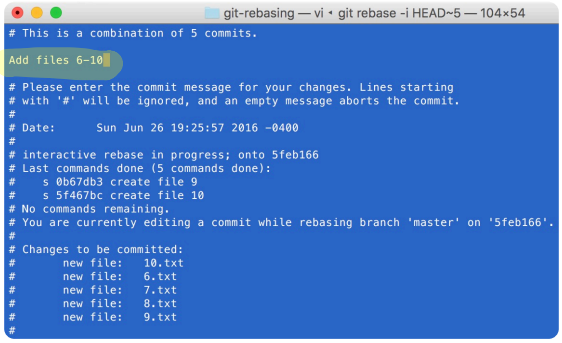
```
for i in 1 2 3 4 5 6 7 8 9 10;
do touch $i.txt;
echo $i > $i.txt;
git add $i.txt;
git commit -m "create file $i";
done
```

```
psm-portable-03:git-rebasing psun$ git lol
* 0190c7c (HEAD -> master) create file 10
* a39fb39 create file 9
* d0604a7 create file 8
* 1148f19 create file 7
* fdbee4a create file 6
* fb20f99 create file 5
* 1cbe13b create file 4
* 73c1c79 create file 3
* bcea292 create file 2
* 2812880 create file 1
```

- **git rebase -i** HEAD~5
  - -i means 'interactive rebasing'
- Replace the last 4 'pick' with 's' for squash. Save.



- Write commit message describing all the squashed commits: "Add files 6-10". Save.



- Admire your newly squashed commits.

```
psm-portable-03:git-rebasing psun$ git lol
* adabf22 (HEAD -> master) Add files 6-10
* 5feb166 create file 5
* 614c46d create file 4
* 3eccf92 create file 3
* 26adad1 create file 2
* 151a388 create file 1
```

Lab 5B - Interactive rebasing: viewing the history of git commands, and resetting to any command

- **git reflog** will give you a history of all the git commands in the repo.

```
psm-portable-03:git-rebasing psun$ git reflog
adabf22 HEAD@{0}: rebase -i (finish): returning to refs/heads/master
adabf22 HEAD@{1}: rebase -i (start): checkout HEAD~5
adabf22 HEAD@{2}: rebase -i (finish): returning to refs/heads/master
adabf22 HEAD@{3}: rebase -i (squash): Add files 6-10
41d85f8 HEAD@{4}: rebase -i (squash): # This is a combination of 4 commits.
113eb7d HEAD@{5}: rebase -i (squash): # This is a combination of 3 commits.
813e2bc HEAD@{6}: rebase -i (squash): # This is a combination of 2 commits.
e07d6fa HEAD@{7}: rebase -i (start): checkout HEAD~5
5f467bc HEAD@{8}: commit: create file 10
0b67db3 HEAD@{9}: commit: create file 9
ea3c6e5 HEAD@{10}: commit: create file 8
c413f62 HEAD@{11}: commit: create file 7
e07d6fa HEAD@{12}: commit: create file 6
5feb166 HEAD@{13}: commit: create file 5
614c46d HEAD@{14}: commit: create file 4
3eccf92 HEAD@{15}: commit: create file 3
26adad1 HEAD@{16}: commit: create file 2
151a388 HEAD@{17}: commit (initial): create file 1
```

- **git checkout** 5f467bc
  - Checking out the last commit that makes sense.
  - You are in 'detached HEAD' state.
- **git checkout -b** branch-with-first-10-commits
  - The new branch now has the commits from before we did the squash.

```
psm-portable-03:git-rebasing psun$ git lol
* 5f467bc (HEAD -> branch-with-first-10-commits) create file 10
* 0b67db3 create file 9
* ea3c6e5 create file 8
* c413f62 create file 7
* e07d6fa create file 6
* 5feb166 create file 5
* 614c46d create file 4
* 3eccf92 create file 3
* 26adad1 create file 2
* 151a388 create file 1
```

Vim Commands

- **esc** to enter command mode
- While in the **command mode**
  - **h** (left) **j** (down a line) **u** (up a line) **l** (right)
    - Move around in command mode
  - **x**
    - Delete
  - **dd**
    - Delete a whole line
  - **dw**, **d4w**
    - Delete a word, delete 4 words
  - **u**
    - Undo
- **:q**
  - Quit
- :**q!**
  - Quit without saving
- **:i**
  - Go into insert mode
- **:wq** or **:x**
  - Save changes and quit

Terminal Commands

- **open .**
  - Open the current folder in Finder.
- **cmd-k**
  - Clear the terminal.
- **mkdir <folder_name>**
  - Make new folder
- **cd <folder_name>**
  - Navigate into the directory with <folder_name>
- **cd ..**
  - Navigate out of current directory.

The most elegantly coded projects to dive into:

- **Javascript**
  - Learn React instead of Angular.
  - **Parse Dashboard**
    - One of the most beautiful React projects
- **Swift**
  - **Alamofire**
    - async requests in swift
  - **Artsy**
    - @ashforrow, @orta
      - Main maintainers of cocoapods
  - **Moya**
    - Network abstraction layer on top of Alamofire
    - @ashforrow
- **html/css**
  - **Bootstrap**
- **languages**
  - **Swift Compiler**

Contributing to open source to is a fantastic way to hone your craft.