Quick notes

- Check Piazza
- Reminders:
  - Assign7 due 29th Nov
  - Assign8 due 4th Dec
  - Weekly quiz
- Assignment 10 starts TODAY!
  - **Email me your topic**
  - Create presentation (<10 min, semi-strict) + brief summary (<200 words)
    - Must present to class (email me if you're in a different timezone)
    - You can pre-record (e.g. youtube)
  - Order is here (tentative), let me know ASAP if you are unable to present.
    - May have to shift up/down depending on time
  - https://docs.google.com/spreadsheets/d/1rcPnnaa2zH8_3fi3qSLgCKRDdwKgqML32HocBnEv6q0/

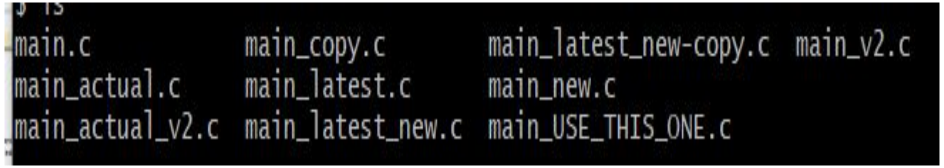
Feedback / Office Hours / Other

- Tameez Latib
  - tameezlatib@gmail.com, please add “CS35L” to the subject line
  - Office Hours: Monday **4pm-6pm** (or by appointment)
  - Feedback: https://forms.gle/6kcJ2aJtzAzFMhHQ7 (anonymous google form)
- If you guys are stressed out:
  - CAPS (https://www.counseling.ucla.edu/)
    - Free with UC ship
- Week 8!
  - We’re almost there...

Git - version control

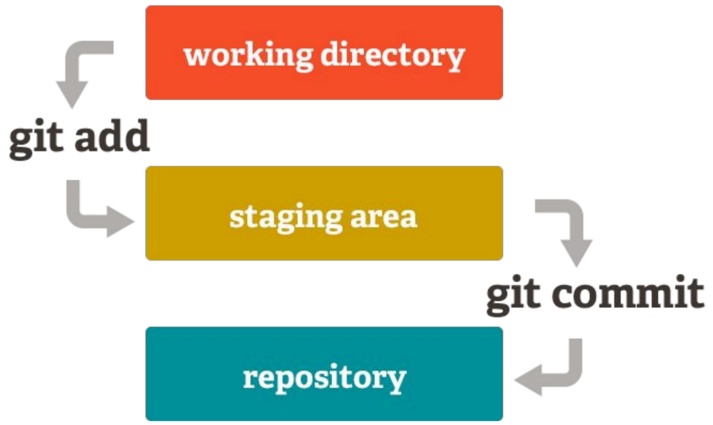
- Why?
  - Revert to previous (working) versions
  - Collaborative (instead of google docs)
- How?
  - Tools: Github / bitbucket / etc
  - Idea: You have local (most recent) version, server stores version history
- Multiple people can work on same codebase
  - You want to change some algorithm, your coworkers wants to change app layout
- History of changes
  - Possible that something in newer version is not stable/working
- Multiple parallel versions
  - Can always merge later,
  - possible that two teams have conflicting code that will be resolved later

Avoid this!



```
$ ls
main.c          main_copy.c        main_latest_new-copy.c  main_v2.c
main_actual.c   main_latest.c      main_new.c
main_actual_v2.c main_latest_new.c main_USE_THIS_ONE.c
```

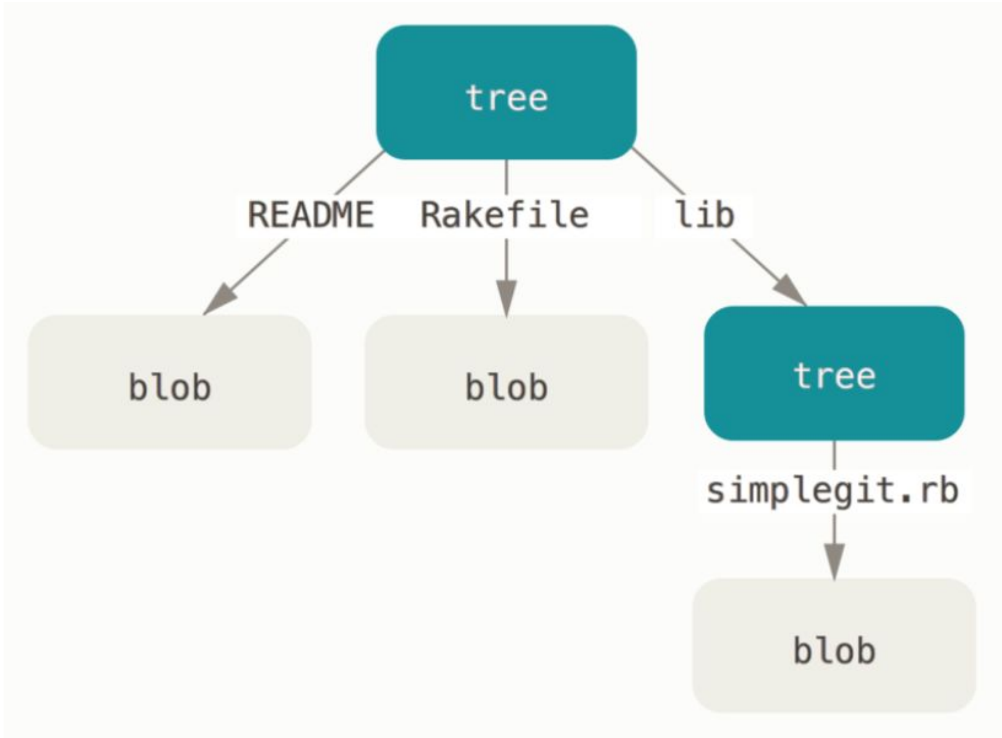
Git - Main idea



- ❖ Working directory is your pc
- ❖ You can selectively add files
  - ➢ I.e. what changes you made that are tested
- ❖ Lastly, you commit to the repository
  - ➢ Repository (repo) is shared
- ❖ Terminology:
  - ➢ Push: you **push** code up **to** shared repo **put latest** (local -> shared repo)
  - ➢ Pull: you **pull** code **from** shared repo to **get latest** (shared -> local repo)
- ❖ In depth explanation of why staging area?
- ❖ https://stackoverflow.com/questions/49228209/whats-the-use-of-the-staging-area-in-git

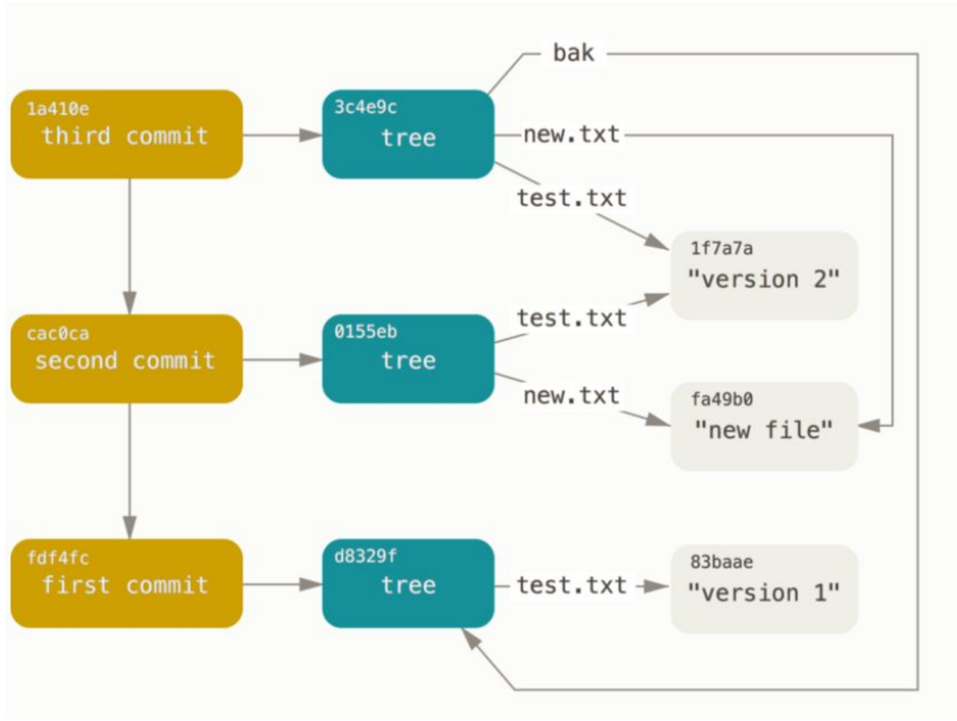
Git - behind the scenes

- Tree:
  - Like a folder
  - Pointer to trees/blobs
- Blob:
  - Like a file
  - Point to contents of file
- Snapshot:
  - Everything to the right
  - Content of repo at some point in time
  - (think screenshot of video)
    - Single [saved] instant in timeline



Git - behind the scenes

- Saves space by using pointers
- Here we have 3 commits
  - 3 snapshots
- Note that each object has a unique Hash / identifier

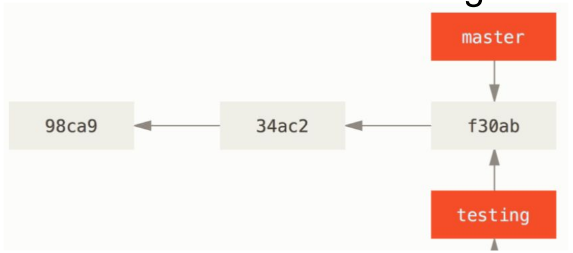


- Reference
- https://git-scm.com/book/en/v2/Git-Internals-Git-Objects (above image from here, has tutorial)

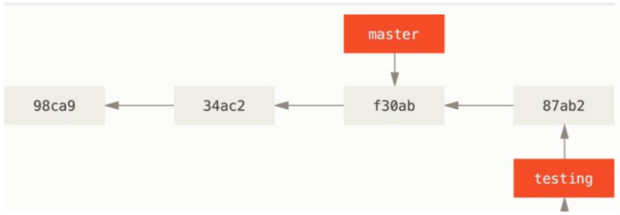
Git - branches

- ❖ Multiple teams work on same codebase, but potentially break each others' code
- ❖ Solution is to have branches (if snapshots are screenshots in a video, branching is similar to having multiple [similar] videos)

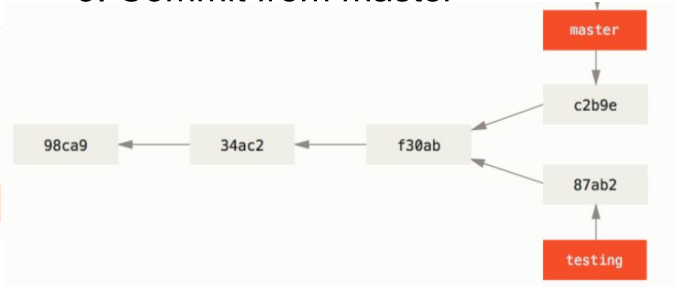
1. Create branch testing



2. Commit from testing



3. Commit from master



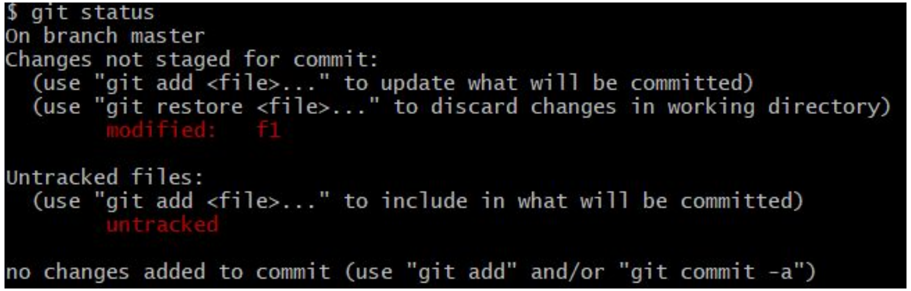
- ❖ Terminology: "master" branch the first created (automatically)
  - ➢ Term is a bit problematic: will be replaced by "main" soon
- ❖ https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Git - guide (summary)

- Git init
  - Initialise directory as git directory
  - Check the .git folder it creates!
- Git clone [url]
  - Copy existing git directory
- **Git add** (put in staging area), **git commit** (create commit/snapshot!)
- **Git push** (push up changes)
- **Git pull**
  - Git fetch (get changes) + git merge (try to merge changes to your copy)
- Git checkout [-b] branchname
  - With -b, creates a new one and switches to branch
  - Without -b, switch to existing branch
  - Can checkout a commit by replacing branchname with commit id

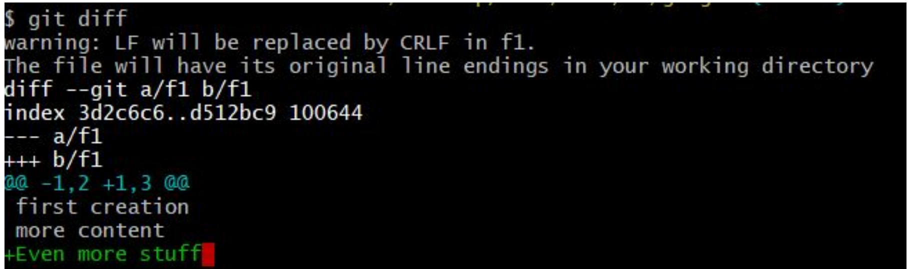
Git - (more stuff)

- Git status
  - Which files are tracked, modified, etc



```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   f1

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        untracked

no changes added to commit (use "git add" and/or "git commit -a")
```

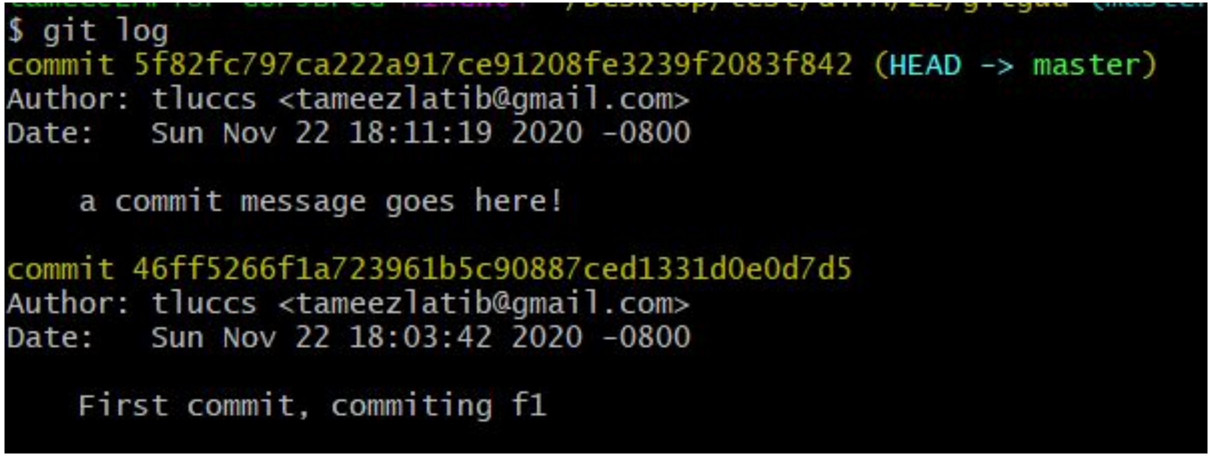
- Git diff
  - Shows all current changes (diff -u) (in working directory, or --stages for staging area)
  - Can also use to compare different commits or different branches



```
$ git diff
warning: LF will be replaced by CRLF in f1.
The file will have its original line endings in your working directory
diff --git a/f1 b/f1
index 3d2c6c6..d512bc9 100644
--- a/f1
+++ b/f1
@@ -1,2 +1,3 @@
 first creation
 more content
+Even more stuff
```

Git - (more stuff)

- ❖ Git log
  - ➢ Shows commit history



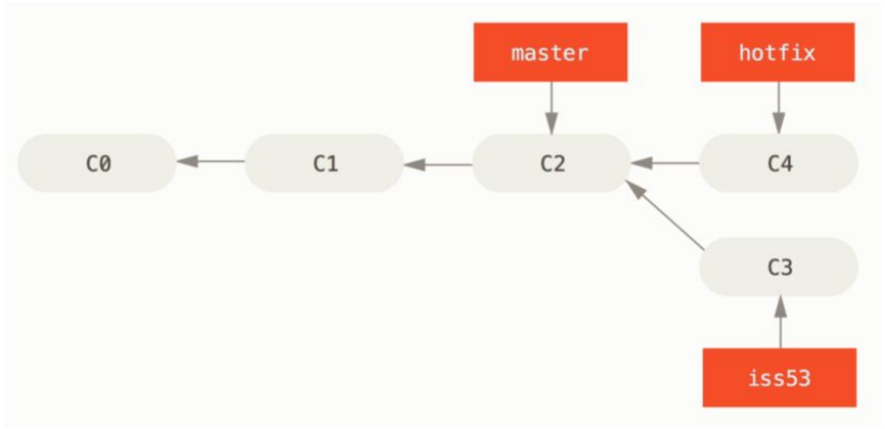
```
$ git log
commit 5f82fc797ca222a917ce91208fe3239f2083f842 (HEAD -> master)
Author: tluccs <tameezlatib@gmail.com>
Date:   Sun Nov 22 18:11:19 2020 -0800

    a commit message goes here!

commit 46ff5266f1a723961b5c90887ced1331d0e0d7d5
Author: tluccs <tameezlatib@gmail.com>
Date:   Sun Nov 22 18:03:42 2020 -0800

    First commit, commiting f1
```

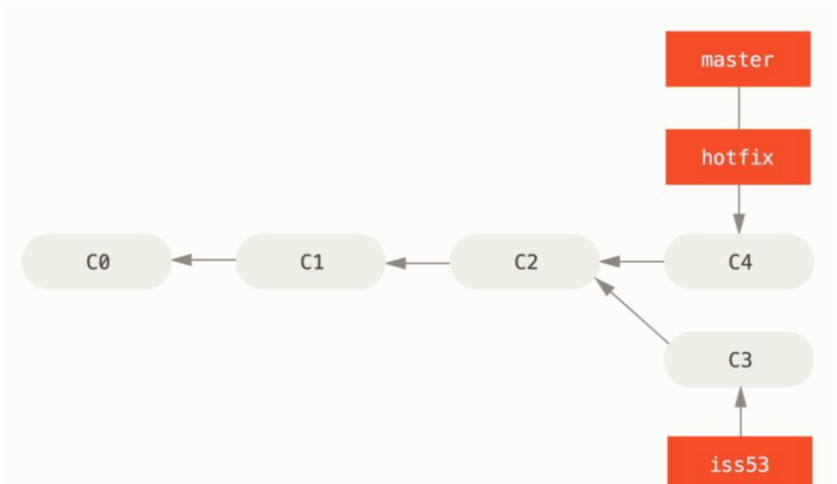
Git - merging

- What to do with two branches?
- Want to combine changes from two branches
- Merge!
- Git checkout branch_1
  - Go into whatever branch you want
- Git merge branch_2
  - Create new commit using branch_1 and branch_2
- Make sure you remember to git add / git commit
- If you want, delete old branch, git branch -d branch_2
- https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

Git - merging, example

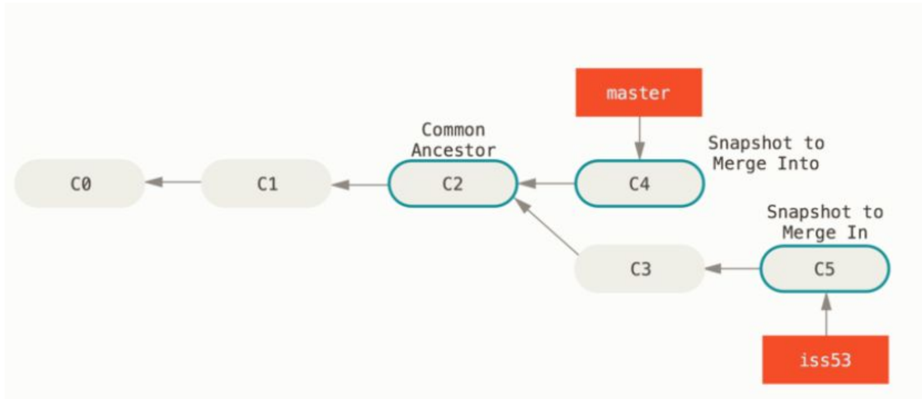
- Currently at this state:
- First, want to merge master to hotfix
  - This just moves the pointer
- How?
- Git checkout master
- Git merge hotfix
- Called fast-forward merge

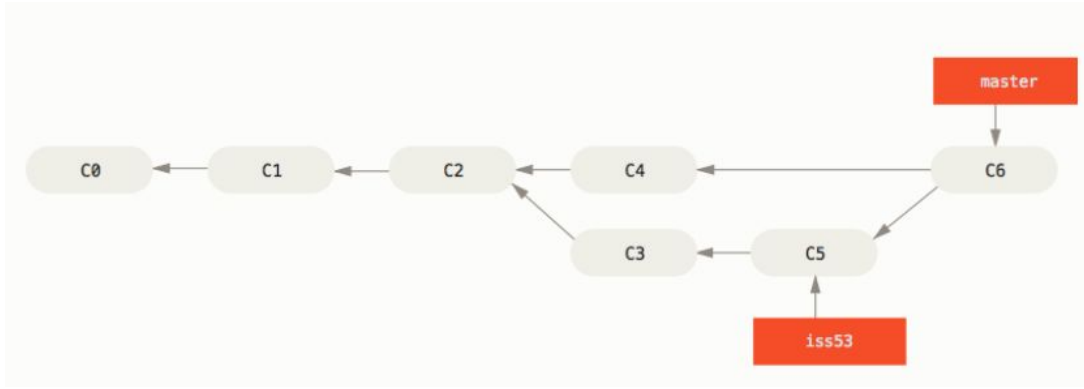




Git - merging, example 2

- Currently at this state:
- Now, want to merge master, iss53
- Note the 3 important snapshots
- Git checkout master
- Git merge iss53
- There will be merge conflicts
  - Same file edited differently
  - x=5 in iss53, x=7 in master

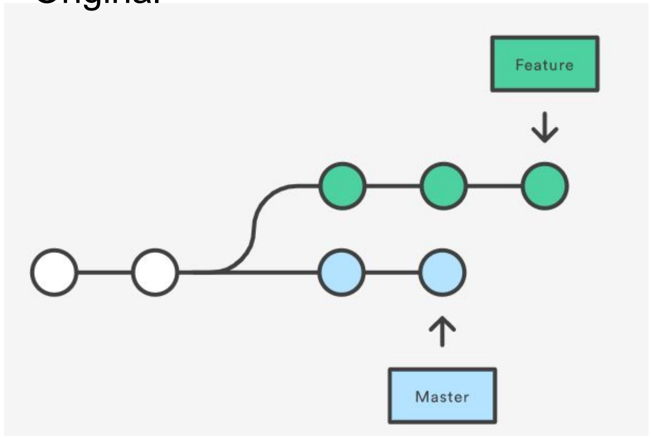




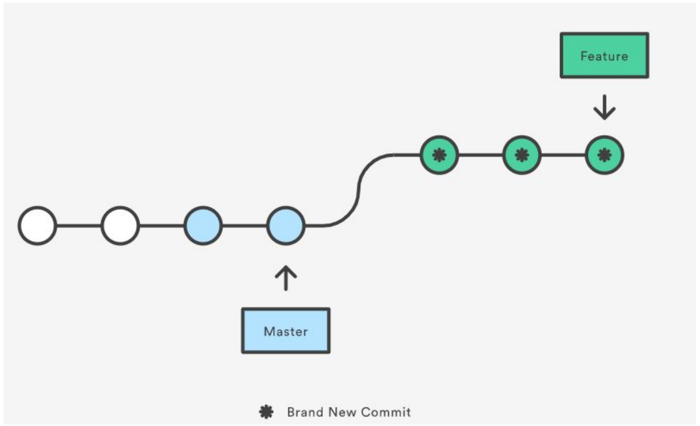
Git - merging vs rebase

- Both want to combine changes from two branches
- Merge tries to merge commits [directly]
- Rebase takes commit history from feature, and tries
  To redo it onto master branch
- https://www.atlassian.com/git/tutorials/merging-vs-rebasing

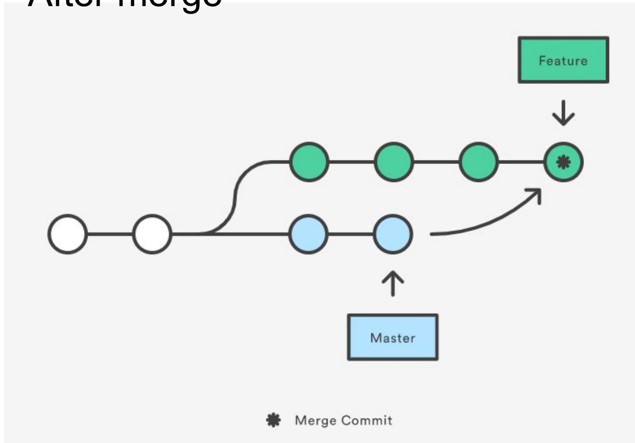
Original



After rebase



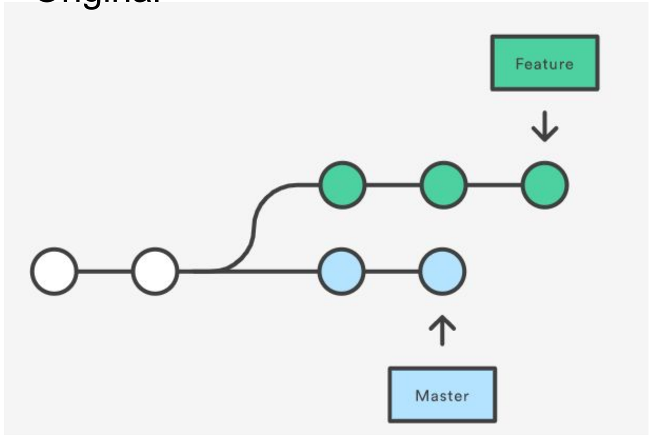
After merge



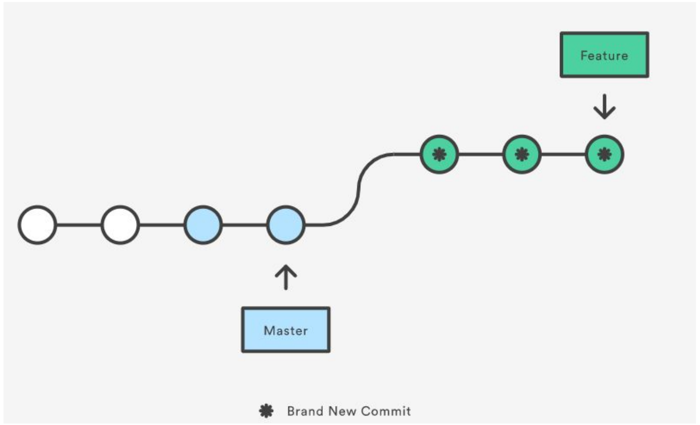
Git - merging vs rebase

- Rebase re-writes history
- So it's nice + linear, but messy work during rebase
- Merge has a messy timeline, but easier to do
- Generally don't rebase to public repos

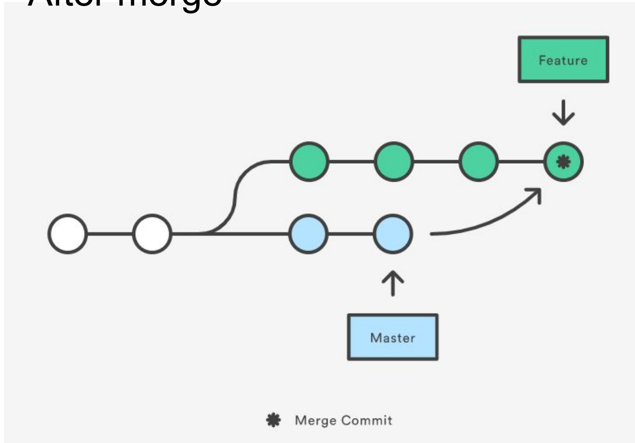
Original



After rebase



After merge



Git - other

- Git clean
  - Remove untracked
- Git restore
  - Revert files to last commit
- Git format-patch [num-prev-commits] [commit id] --stdout > patch
  - Git format-patch -2 --stdout will produce diff from your current commit to 2 commits ago
- Git am < patch
  - Apply patch
- Create a .gitignore file
  - Specifies which files to completely ignore

Git - with emacs

- https://www.gnu.org/software/emacs/manual/html_node/emacs/Version-Control.html
- Emacs version control
- Vc-diff: compare current files vs what you pulled
- Vc-revert: vc-diff + check if you want to revert files
- Check out emacs diff mode
- https://www.gnu.org/software/emacs/manual/html_node/emacs/Diff-Mode.html

Questions??