hackerschool
by NUS Hackers

Introduction to Git

Time to Git Gud

Raynold Ng Yi Chong

8 September 2018

Where are we?

NUS Hackers

http://nushackers.org

Hackerschool
Friday **Hacks**
Hack & Roll
NUS **Hacker**space

About Me

Hi! I am Raynold. My github is
`https://github.com/raynoldng`
A Year 3 Computer Science Undergraduate who loves building stuff.

Have been doing web development for the past 2 years.
Interests: algorithms and math

About This Workshop

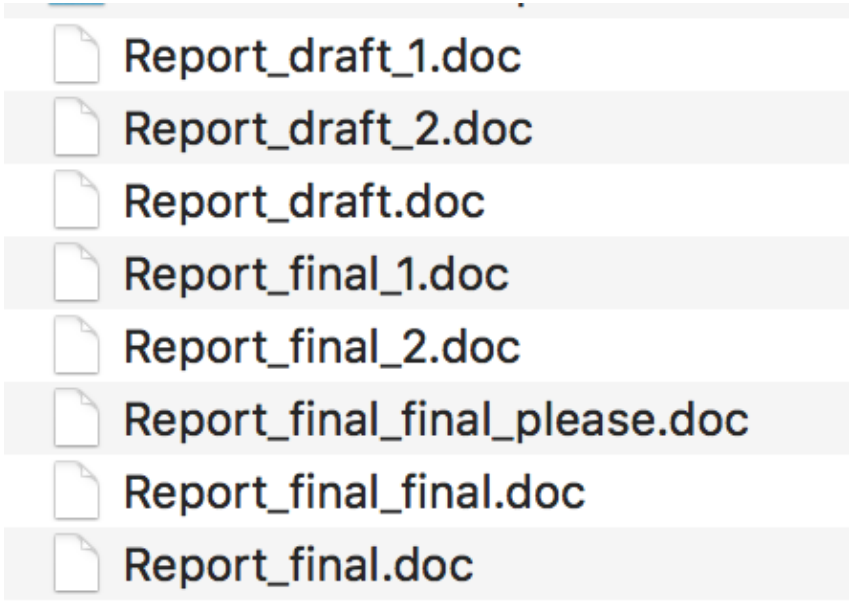
- Beginner level workshop
- No prior knowledge assumed
- Basic and advanced features of Git
- Better manage your code base and collaborate with others

Table of Contents

Required Software

- **git** (`https://git-scm.com/downloads`)
- **VS Code** (`https://code.visualstudio.com/`) or your favorite text editor

Have you ever seen:



What is version control

- Category of software tools that help software team manage checks changes to source code over time
- every modification to code tracked in a special kind of database
- version control software (VSC) essential part of modern software team's professional practices
- Example: `https://github.com/torvalds/linux/commits/master`

What is git?

- Most widely used modern VSC
- Originally developed in 2005 by Linus Torvalds, famous creator of Linux operating system kernel
- Pros: Performance, Security, Flexibility
- Cons: Hard to learn???
- Download it here: `https://git-scm.com/downloads`

Setting up git

Set your user name and email. Every git commit uses this information and baked into your commits. `--global` option so that git will always use that information on that system

```
1 git config --global user.name <your name>
2 git config --global user.email <your email address>
3 git config -- global --add color.ui true
```

Where are we?

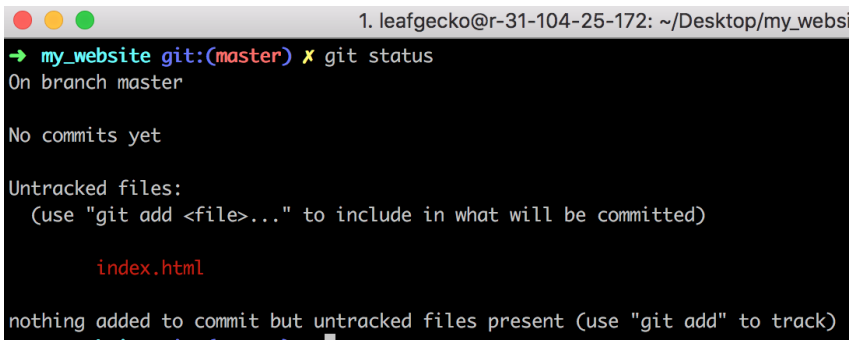
Create a local git repo

- When creating a new project on your local machine, you'll first create a new repository
- Enter the following into the terminal

```
cd ~/Desktop
mkdir my_site
cd my_site
git init
```

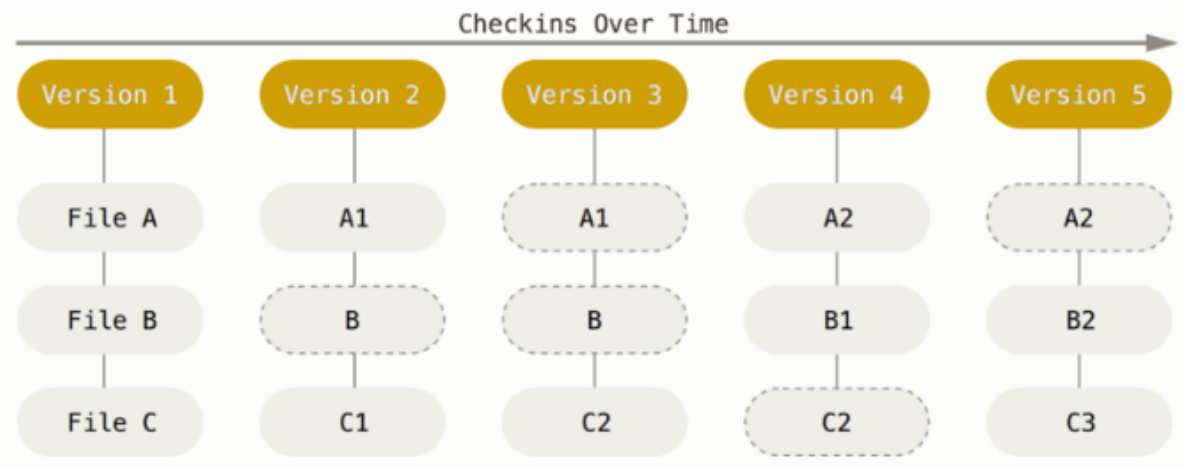
Add a new file to the repo

- We are going to reuse the website created from last week. If you don't have it, get it here
- once you've added or modified files, in the repo folder, git will notice changes made in the repo
- use the `git status` to see which files git knows exist



Git Basics

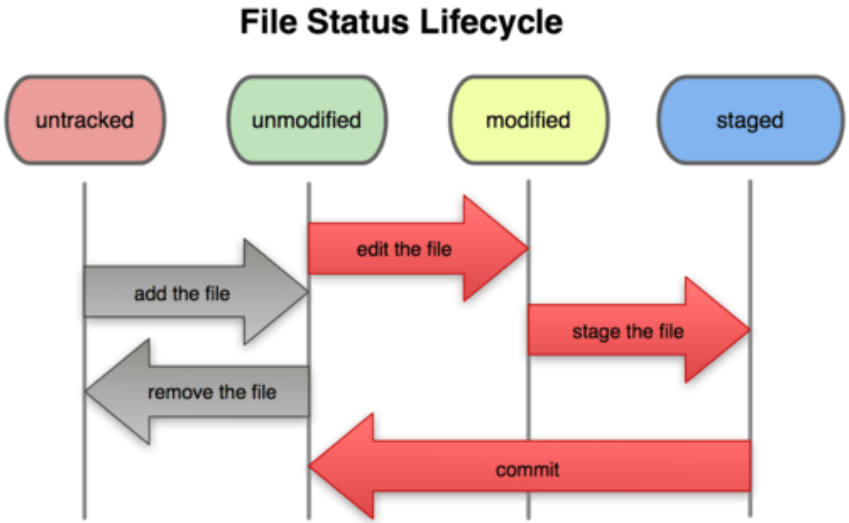
- git all about snapshots, not deltas
- git thinks of its data as a series of snapshots of a miniature filesystem
- every time you commit, git takes a photo of what your files look like and stores a reference to that object
- git data is a stream of snapshots

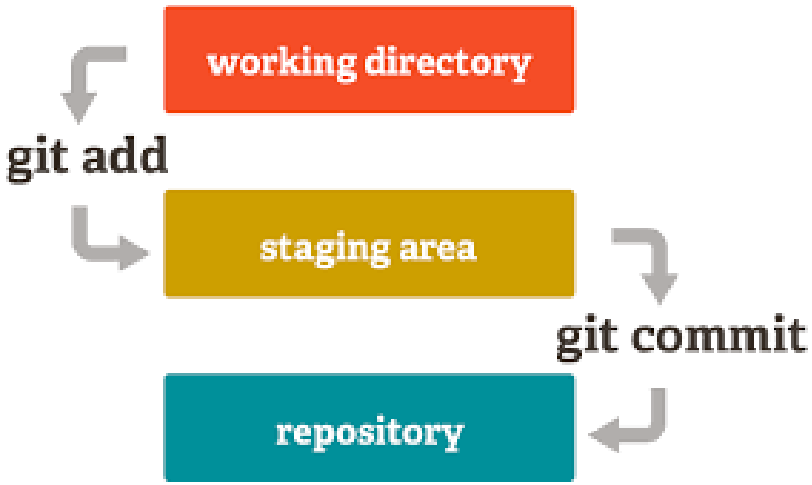


Master the Three States (Elements)

- **commit**: record of what files you have changed since the last commit
- files in your git repository can be in three main states:
  - **untracked**: any files that are not in your last snap shot and not in your staging area
  - **unmodified**: files not modified since last snap shot
  - **committed**: data is stored safely in your local database
  - **modified**: file changed but have not committed to your database yet
  - **staged**: modified file marked in its current version to go into next commit snapshot

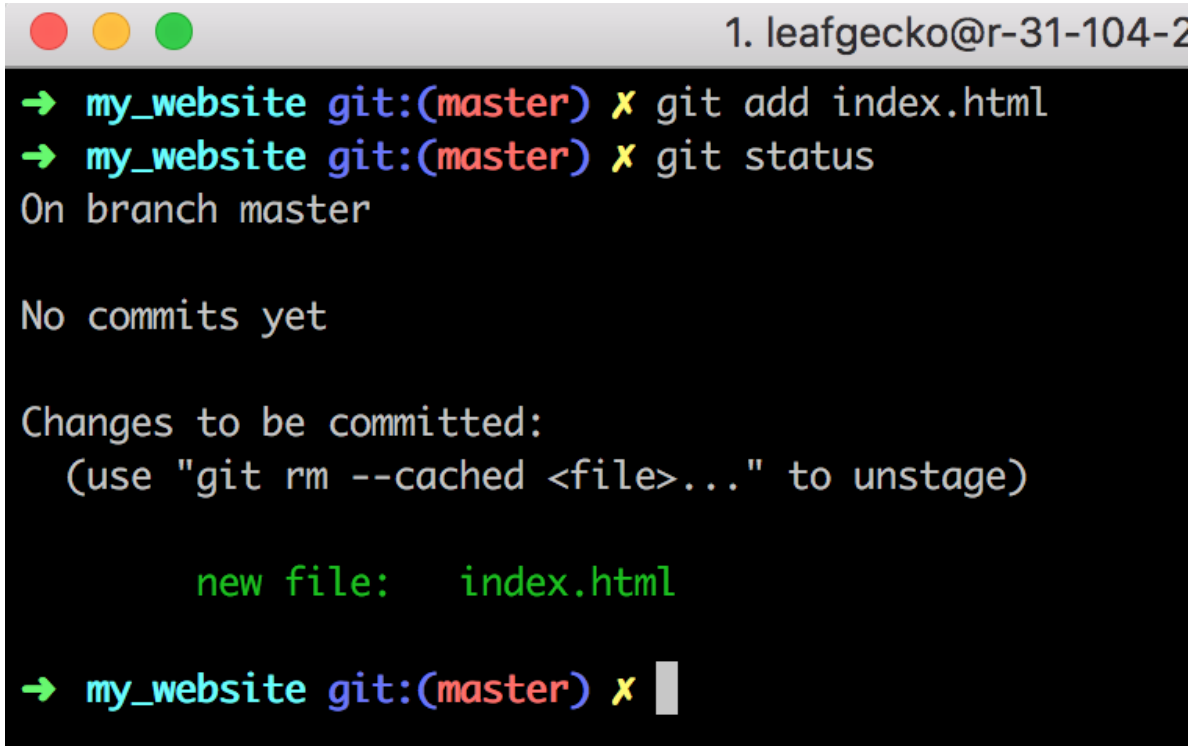
git workflow





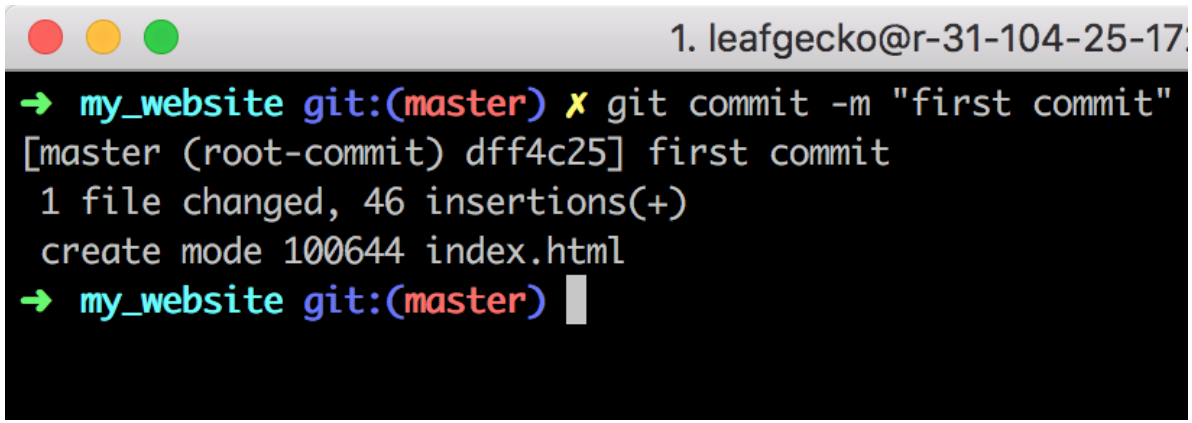
Add a file to staging environment

- add a file to staging with the `git add <file>` command
- the after `git add`, the file has **not** yet been added to a commit



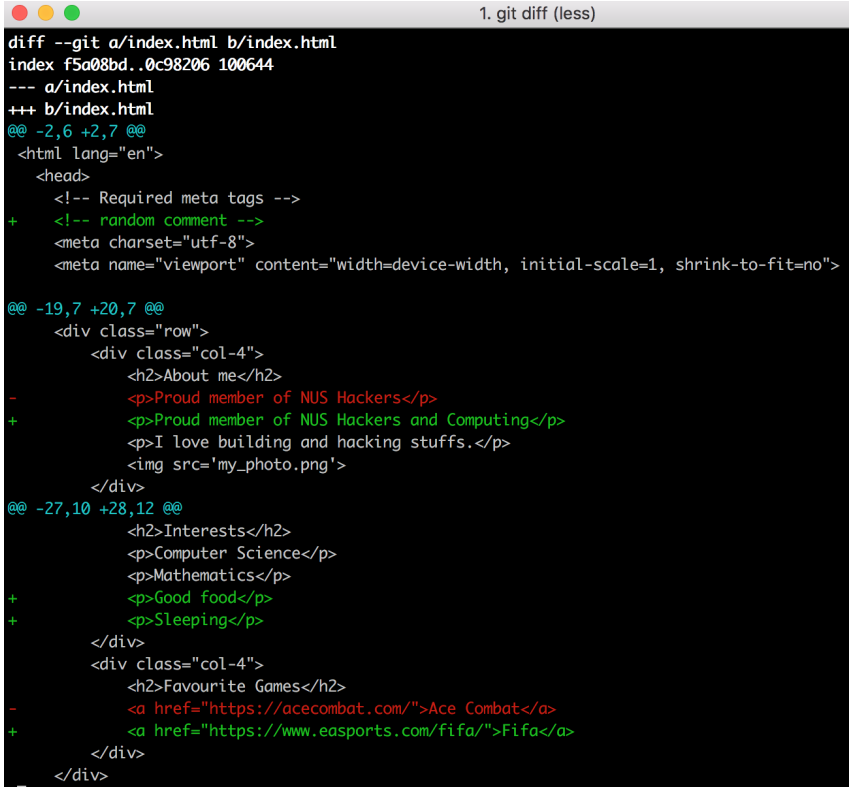
Create a commit

- Run the command `git commit -m "Your message to commit"`
- The message should be related to the commit



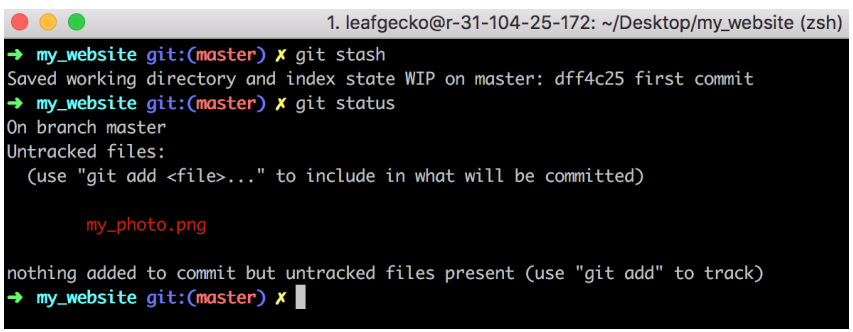
Comparing changes with git diff

- Diffing is a function that takes two input data sets and outputs changes between them
- Add/delete/edit some lines in `index.html` and run `git diff` to show any uncommited since last commit
- `git diff` used to show changes between commits, commit and working tree etc. See `https://git-scm.com/docs/git-diffdocumentation`



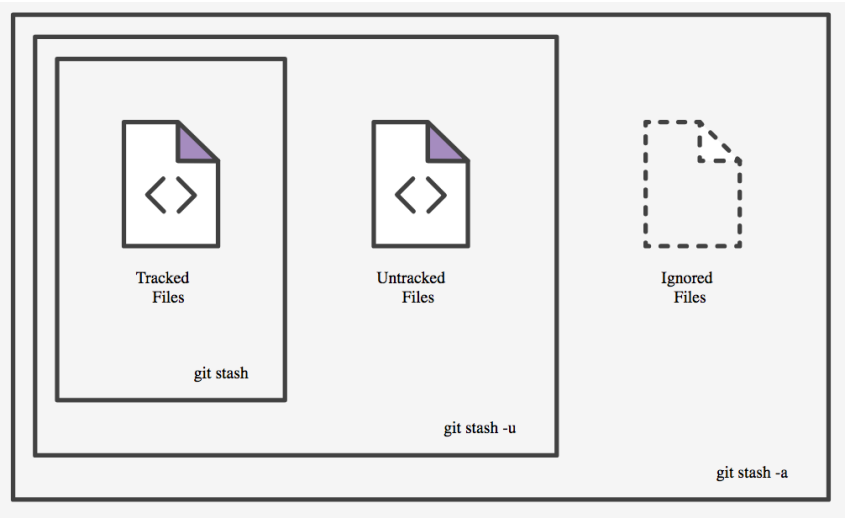
Stashing changes

- stashing is handy if you need to quickly switch context and work on something else
- `git stash` takes your uncommitted changes (both staged and unstaged) and saves them away for later use
- By default, `git stash` will not stash new files and files that are ignored(!), add `-u` or `--include-untracked` to stash untracked files



Stashing untracked or ignored files

- By default, `git stash` will not stash new files and files that are ignored(!), add `-u` or `--include-untracked` to stash untracked files
- annotate your stash with a description: `git stash save "message"`



Applying Stash

- Reapply stashed changes with `git stash pop`
- popping **removes** the changes from your stash and reapplies them to your working copy
- `git stash apply` to reapply changes to your working copy and keep them, useful if want to apply on multiple branches(!)

.gitignore

- Ignored files are usually build artifacts and machine generated files that can be derived from your repository
- common examples:
  - dependency caches like `/node_modules` or /packages
  - compiled code, such as .o, .pyc, and .class files
  - build output directories, such as /bin, /out, or /target
  - files generated at runtime, such as .log, .lock, or .tmp
  - personal config files like `.idea/workspace`

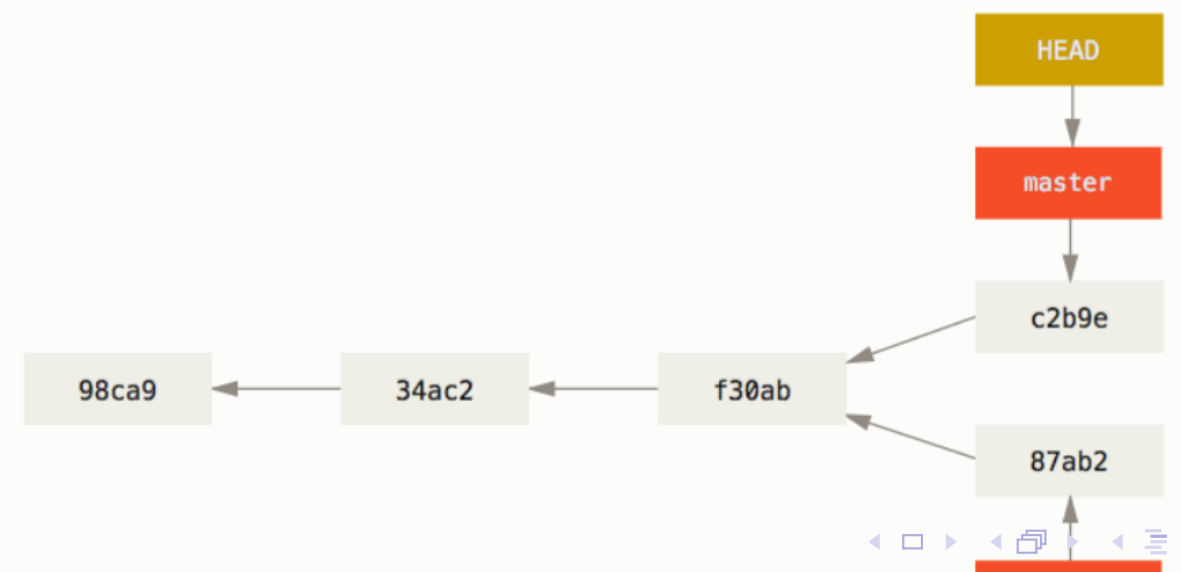
Git ignore patterns

- **/logs
- **/logs/debug.log
- *.log
- /debug.log
- debug.log
- See the full list here

Where are we?

Branching

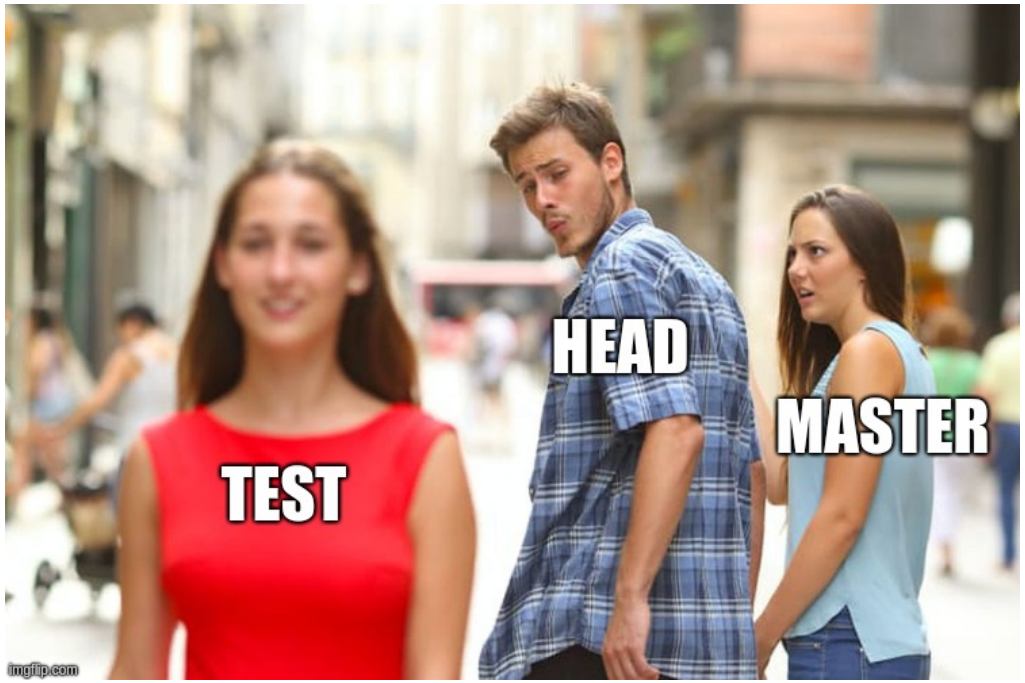
- allow you to diverge from the main line of development and continue work without messing with the main line
- killer feature of git as it is incredibly fast and lightweight
- a branch is a lightweight pointer to a commit, default pointer is `master`
- `HEAD`: special pointer to local branch that you are currently on



Common branch commands

- `git branch`: list all branches
- `git branch <branch>`: create a new branch of given name
- `git branch -d <branch>`: delete specified branch, cannot delete if have unmerged changes
- `git branch -D <branch>`: force delete specified branch
- `git branch -m <branch>`: rename current branch to <branch>
- `git branch -a`: list all remote branches

Git checkout



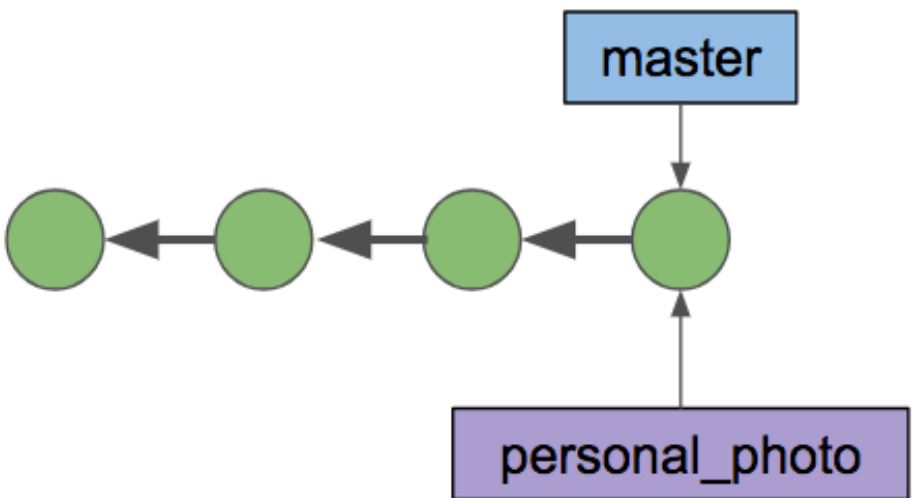
Creating and checking out a branch

- checking out is the act of switching between different versions of a target entity
- entities: files, commits, and branches

```
1 git branch personal_photo
2 git checkout personal_photo
```

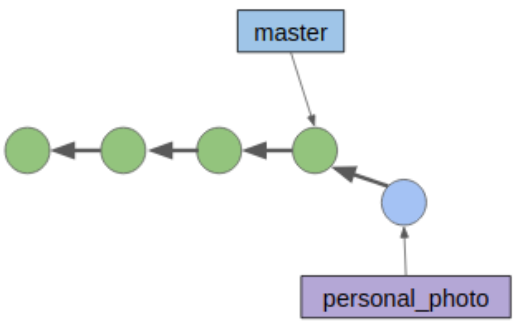
OR:

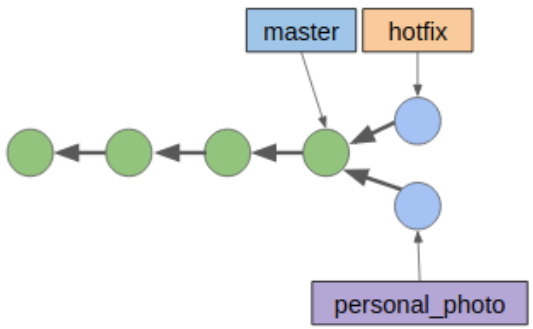
```
1 git checkout -b personal_photo
```

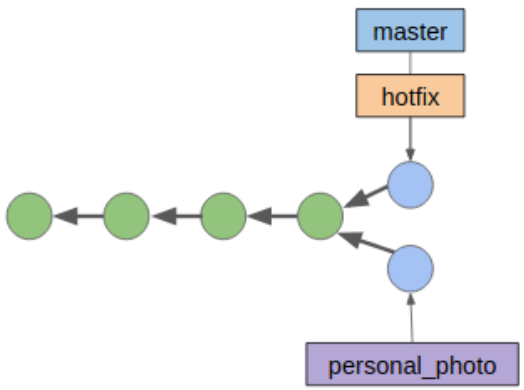


Branching Workflow

1. add photo to `index.html` and commit it, doing so moves `personal_photo` forward
2. urgent fix: change the name, create a `hot_fix` branch, once done merge it back into `master`
3. switch back to adding your photo and merge it back into master when done







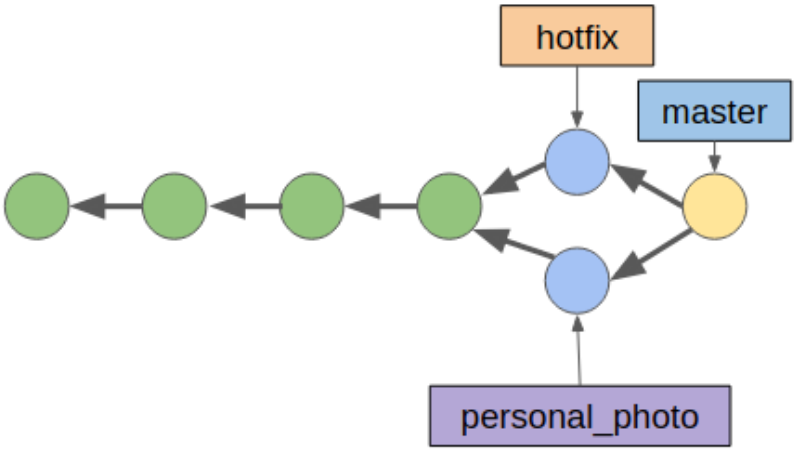
git fast forwarding

- git does a fast forward when you merge a branch that is ahead of your checked out branch (e.g. merge `hotfix` into `master`
- both branches point the same commit and no new commit is made

```
test@julius-ThinkPad-X220: ~/Desktop/r
File  Edit  View  Search  Terminal  Help
test@julius-ThinkPad-X220:~/Desktop/my_website$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
test@julius-ThinkPad-X220:~/Desktop/my_website$ git merge hotfix
Updating dff4c25..c09c6be
Fast-forward
 index.html | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
```

Three way merge

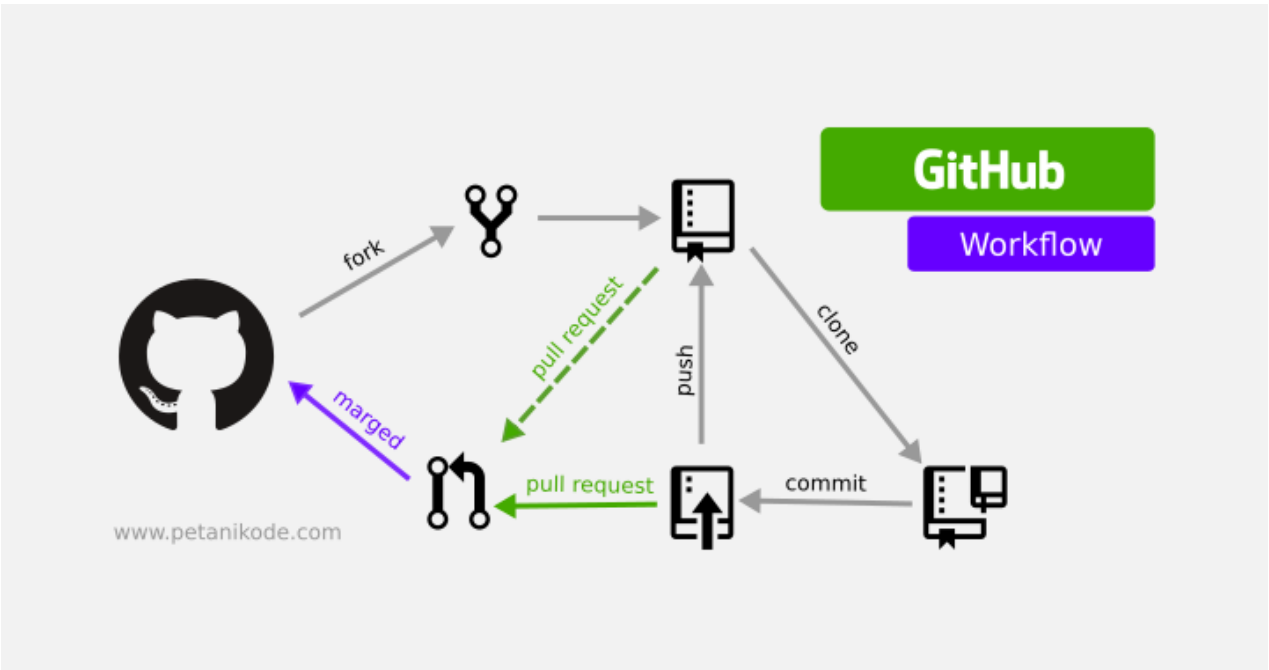
- however a three way merge is not possible if branches have diverged
- git has to do a 3 way merge, a dedicated commit is used to tie together the two histories
- 3 way: three commits to generate the merge commit: two branch tips and their common ancestor
- `git log --oneline --decorate --graph --all`




```
test@julius-ThinkPad-X220:~/Desktop/my_website$ git merge personal_photo
Merge made by the 'recursive' strategy.
 photo.png | Bin 0 -> 71851 bytes
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 photo.png
test@julius-ThinkPad-X220:~/Desktop/my_website$ git log --oneline --decora
te --graph --all
*   c56b131 (HEAD -> master) Merge branch 'personal_photo'
|\
| * e3fa5d3 (personal_photo) add my photo
* | c09c6be (hotfix) trivial fix
|/
* dff4c25 (origin/master, origin/HEAD) first commit
```

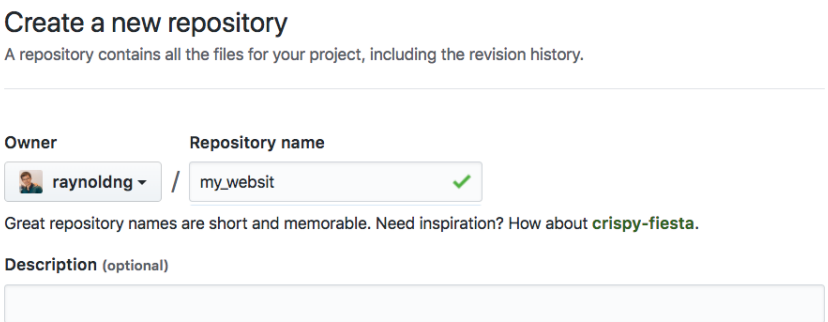
What is Github?

- code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere
- alternative: bitbucket



Create your github repo

1. Create a github account if you haven't done so already
2. Create a new repository `my website` or any name you want
3. Push your code to this repo




```
1  git remote add origin
   ↪  <url>
2  git push -u origin
   ↪  master
```

Working with remotes

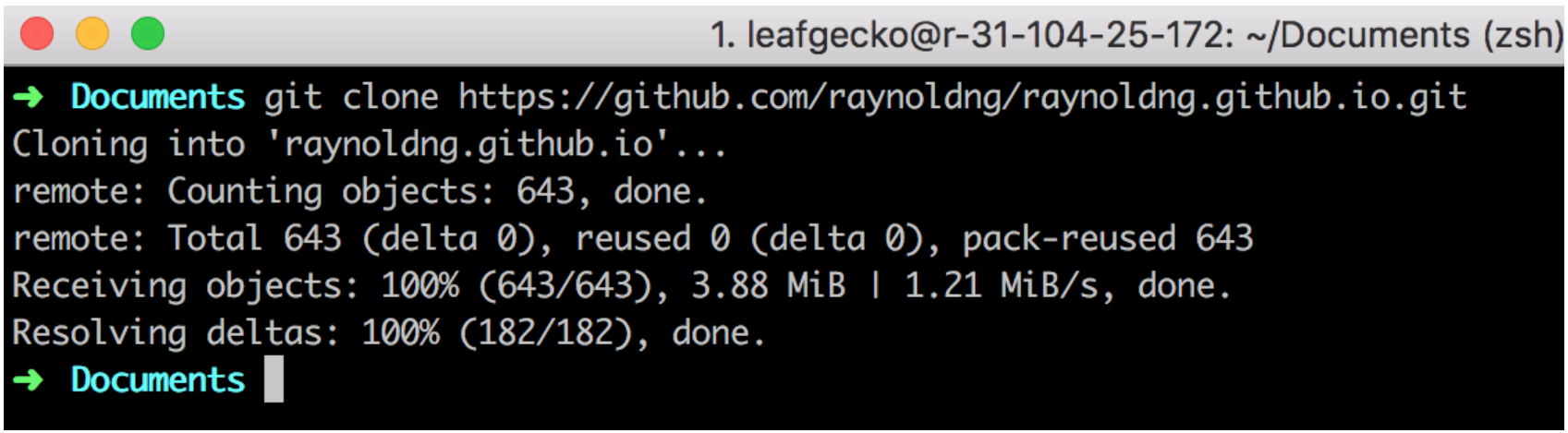
- remote repos are versions of your project that are hosted on the Internet (Github) or somewhere
- collaborating with others involves managing these remote repositories and pushing and pulling data between them

Fetching, Pushing and Pulling

- `git fetch <remote>`: goes to remote project and pulls down all the data from that remote project that you don't have yet
- `git pull <remote>`: fetch and merge that remote branch into your current branch
- `git push <remote> <branch>`: push branch to remote project, you need write permissions to that remote project

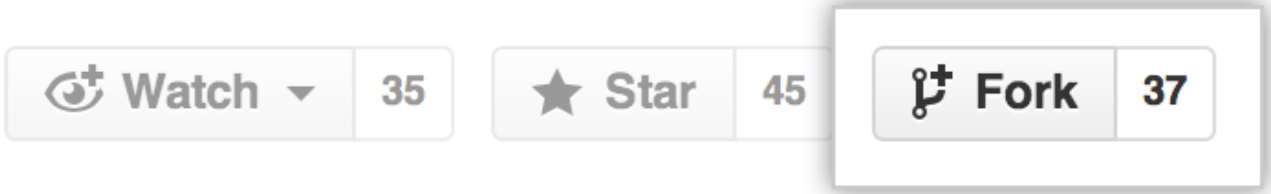
Cloning a repo

- `git clone` target an existing repo and create a clone, or copy of the target repository
- cloning automatically creates a remote connection called origin pointing back to the original repository



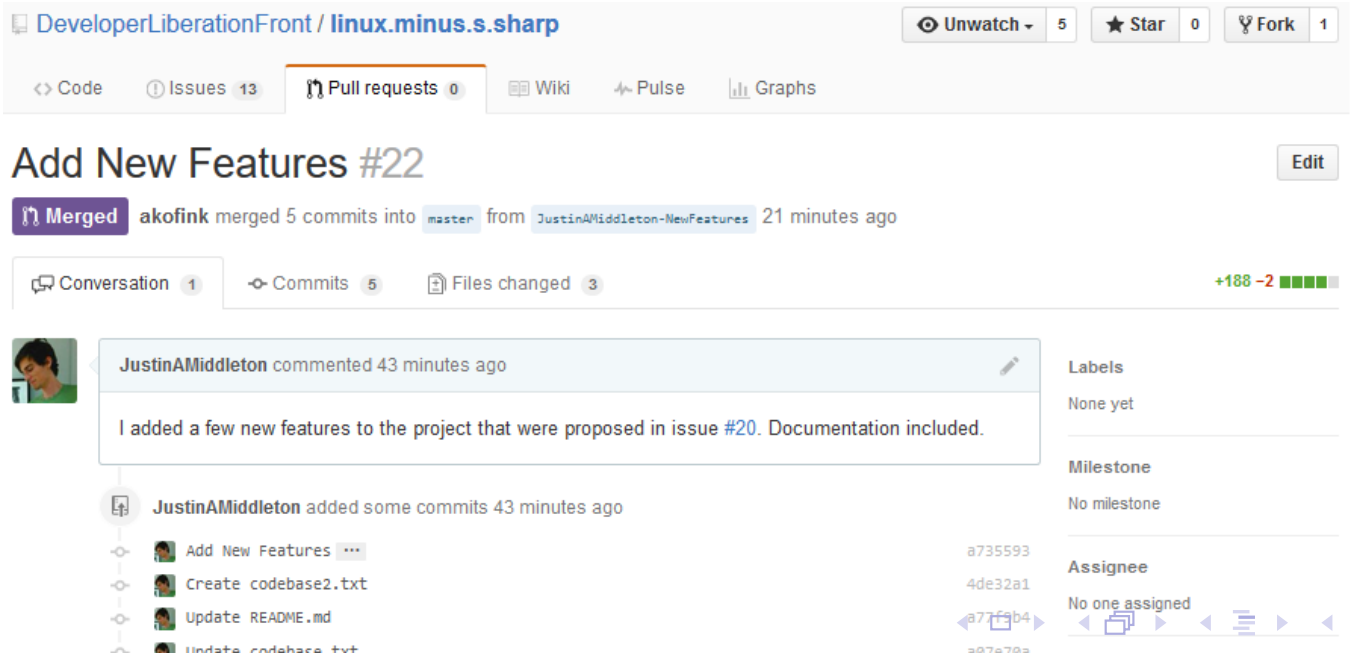
Forking a repo

- forking produces a personal copy of someone else's project
- acts as the bridge between original repository and your personal copy
- you can submit pull requests to help make other people's project better



Making a Pull Request

- mechanism for a developer to notify team members that they have completed a feature
- once feature is ready, the dev files a pull request via their Github account
- pull request is more than just a notification—it's a dedicated forum for discussing the proposed feature



Pair Activity

1. Learn one interesting fact about the person sitting next to you
2. Fork his/her project and create a branch `fun_facts` and add the fun fact under the `About Me` section
3. Create a pull request
4. Accept your neighbor's pull request

Where are we?

Where are we?

Talk to us!

- **Feedback form**: `https://tinyurl.com/hs2018-html`
- **Completed:**
  - HTML/CSS
  - Git
- **Upcoming hackerschool**:
  - HTML/CSS practice
  - Introduction to ES6