hackerschool
by NUS Hackers

Introduction to Git

sildes: `https://tinyurl.com/hs-2018-git-slides`

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

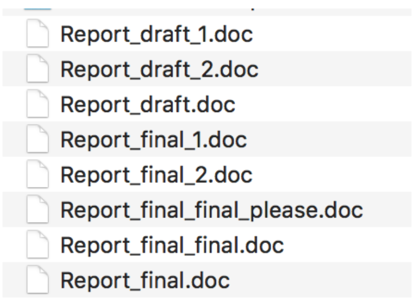
- Beginner level workshop
- No prior knowledge assumed
- Basic and advanced features of Git
- Better manage your code base and collaborate with others

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:


Report_draft_1.doc
Report_draft_2.doc
Report_draft.doc
Report_final_1.doc
Report_final_2.doc
Report_final_final_please.doc
Report_final_final.doc
Report_final.doc

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 (VCS) essential part of modern software team's professional practices
- Example: `https://github.com/torvalds/linux/commits/master`

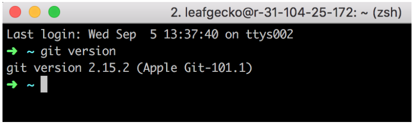
git

mercurial

SUBVERSION

What is git?

- Most widely used modern VCS
- 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`



```
Last login: Wed Sep  5 13:37:40 on ttys002
➜ ~ git version
git version 2.15.2 (Apple Git-101.1)
➜ ~
```

Setting up git

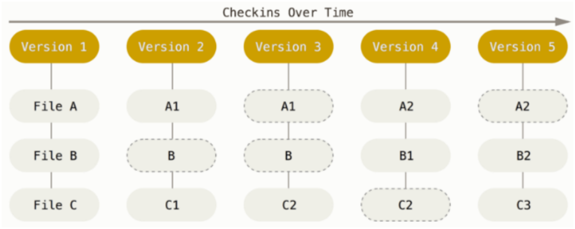
Set your user name and email. Every git commit uses this information and baked into your commits. `--global` option so that git will always use that information on that system

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

Where are we?

Git Basics

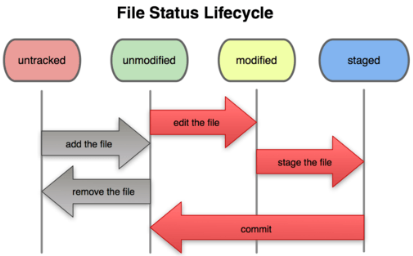
- git all about snapshots, not deltas
- every time you commit, git takes a photo of what your files look like and stores a reference to that object

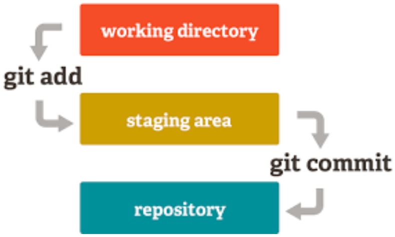


Master the Three States (Elements)

- **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





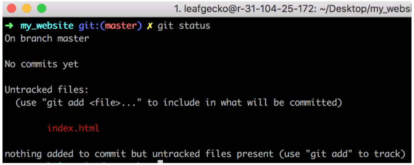
Create a local git repo

- 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

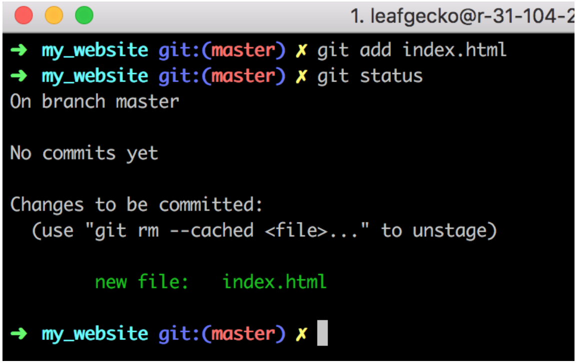
- We are going to reuse the website created from last week. If you don't have it, get it here
- once you've added or modified files, in the repo folder, git will notice changes made in the repo
- use the `git status` to see which files git knows exist



```
➜ my_website git:(master) ✗ git status
On branch master

No commits yet

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

        index.html

nothing added to commit but untracked files present (use "git add" to track)
```

Add a file to staging environment

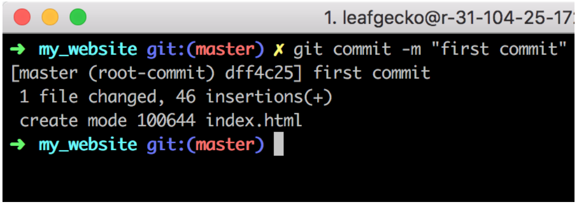
- add a file to staging with the `git add <file>` command
- the after `git add`, the file has **not** yet been added to a commit



```
➜ my_website git:(master) ✗ git add index.html
➜ my_website git:(master) ✗ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   index.html

➜ my_website git:(master) ✗ 
```

Create a commit

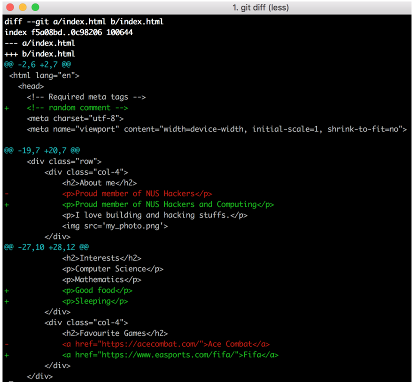
- Run the command `git commit -m "Your message to commit"`
- The message should be related to the commit



```
1. leafgecko@r-31-104-25-17
➜ my_website git:(master) ✗ git commit -m "first commit"
[master (root-commit) dff4c25] first commit
 1 file changed, 46 insertions(+)
 create mode 100644 index.html
➜ my_website git:(master)
```

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-diff`documentation



```
diff --git a/index.html b/index.html
index f5a08bd..0c98206 100644
--- a/index.html
+++ b/index.html
@@ -2,6 +2,7 @@
 <html lang="en">
   <head>
     <!-- Required meta tags -->
+    <!-- random comment -->
     <meta charset="utf-8">
     <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

@@ -19,7 +20,7 @@
     <div class="row">
         <div class="col-4">
             <h2>About me</h2>
-            <p>Proud member of NUS Hackers</p>
+            <p>Proud member of NUS Hackers and Computing</p>
             <p>I love building and hacking stuffs.</p>
             <img src='my_photo.png'>
         </div>
@@ -27,10 +28,12 @@
             <h2>Interests</h2>
             <p>Computer Science</p>
             <p>Mathematics</p>
+            <p>Good food</p>
+            <p>Sleeping</p>
         </div>
         <div class="col-4">
             <h2>Favourite Games</h2>
-            <a href="https://acecombat.com/">Ace Combat</a>
+            <a href="https://www.easports.com/fifa/">Fifa</a>
         </div>
     </div>
```

.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`

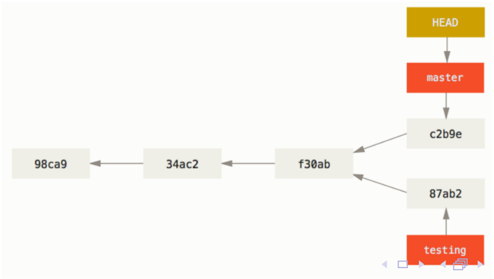
Git ignore patterns

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

Where are we?

Branching

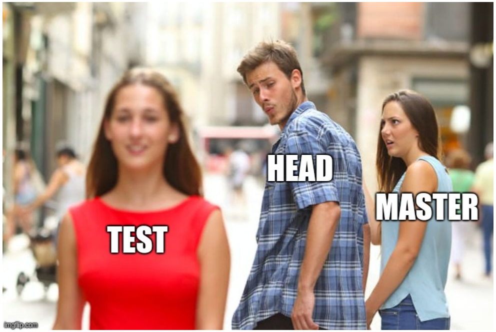
- allow you to diverge from the main line of development and continue work without messing with the main line
- killer feature of git as it is incredibly fast and lightweight
- a branch is a lightweight pointer to a commit, default pointer is `master`
- `HEAD`: special pointer to local branch that you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



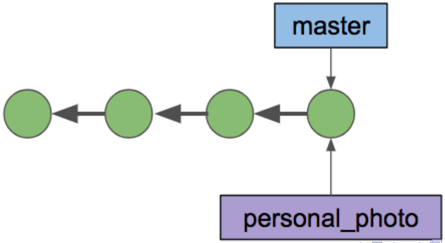
Creating and checking out a branch

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

```
git branch personal_photo
git checkout personal_photo
```

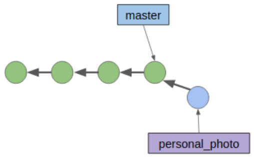
OR:

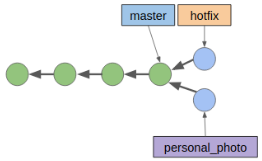
```
git checkout -b personal_photo
```

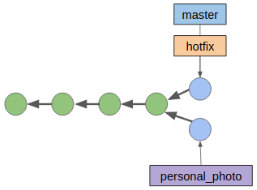


Branching Workflow

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

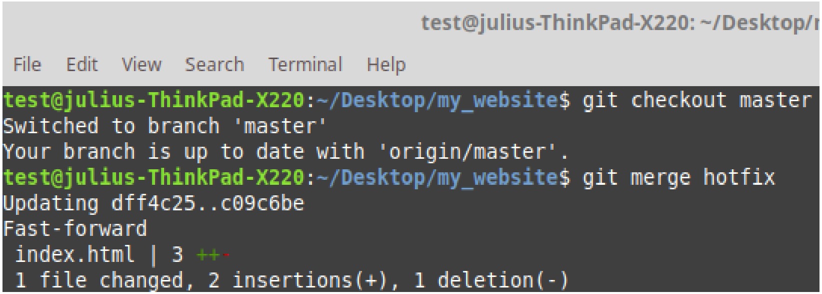






git fast forwarding

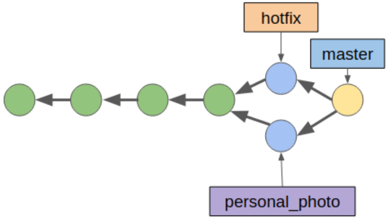
- git does a fast forward when you merge a branch that is ahead of your checked out branch (e.g. merge `hotfix` into `master`
- both branches point the same commit and no new commit is made

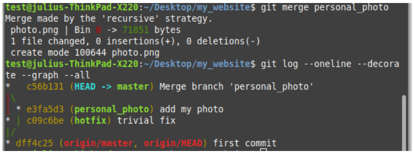

test@julius-ThinkPad-X220: ~/Desktop/
File Edit View Search Terminal Help

```
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`

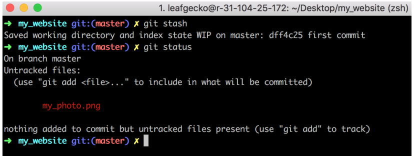




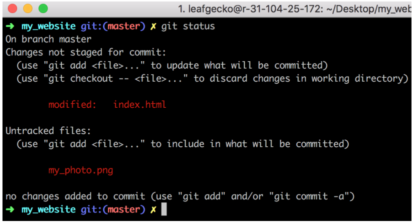
```
test@julius-ThinkPad-X220:~/Desktop/my_website$ git merge personal_photo
Merge made by the 'recursive' strategy.
 photo.png | Bin 0 -> 71851 bytes
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 photo.png
test@julius-ThinkPad-X220:~/Desktop/my_website$ git log --oneline --decora
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
```

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



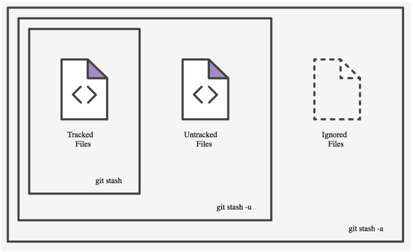
```
1. leafgecko@r-31-104-25-172: ~/Desktop/my_website (zsh)
➜ my_website git:(master) ✗ git stash
Saved working directory and index state WIP on master: dff4c25 first commit
➜ my_website git:(master) ✗ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        my_photo.png

nothing added to commit but untracked files present (use "git add" to track)
➜ my_website git:(master) ✗
```



```
1. leafgecko@r-31-104-25-172: ~/Desktop/my_web
➜ my_website git:(master) ✗ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   index.html

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

        my_photo.png

no changes added to commit (use "git add" and/or "git commit -a")
➜ my_website git:(master) ✗
```

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"`

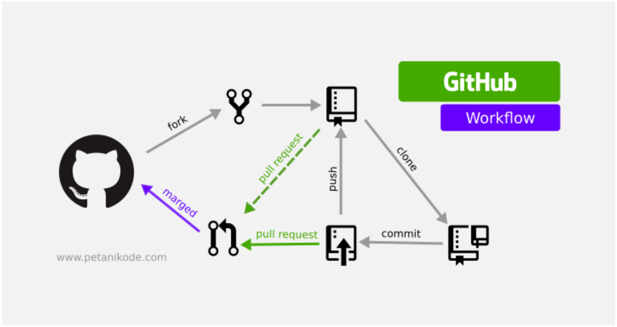


Applying Stash

- 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(!)

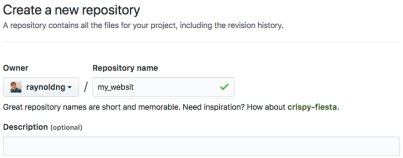
What is Github?

- 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


Create a new repository
A repository contains all the files for your project, including the revision history.

Owner: raynoldng / Repository name: my_websit

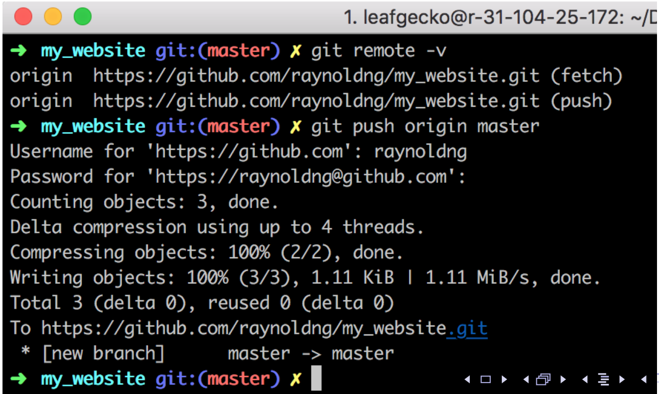
Great repository names are short and memorable. Need inspiration? How about crispy-fiesta.

Description (optional)

```
git remote add origin
↪ <url>
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



1. leafgecko@r-31-104-25-172: ~/D

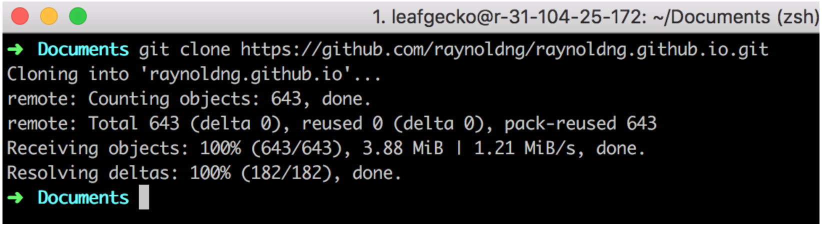
```
➜ my_website git:(master) ✗ git remote -v
origin  https://github.com/raynoldng/my_website.git (fetch)
origin  https://github.com/raynoldng/my_website.git (push)
➜ my_website git:(master) ✗ git push origin master
Username for 'https://github.com': raynoldng
Password for 'https://raynoldng@github.com':
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 1.11 KiB | 1.11 MiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/raynoldng/my_website.git
 * [new branch]      master -> master
➜ my_website git:(master) ✗
```

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

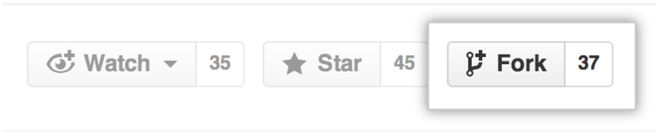


1. leafgecko@r-31-104-25-172: ~/Documents (zsh)

```
➜  Documents git clone https://github.com/raynoldng/raynoldng.github.io.git
Cloning into 'raynoldng.github.io'...
remote: Counting objects: 643, done.
remote: Total 643 (delta 0), reused 0 (delta 0), pack-reused 643
Receiving objects: 100% (643/643), 3.88 MiB | 1.21 MiB/s, done.
Resolving deltas: 100% (182/182), done.
➜  Documents
```

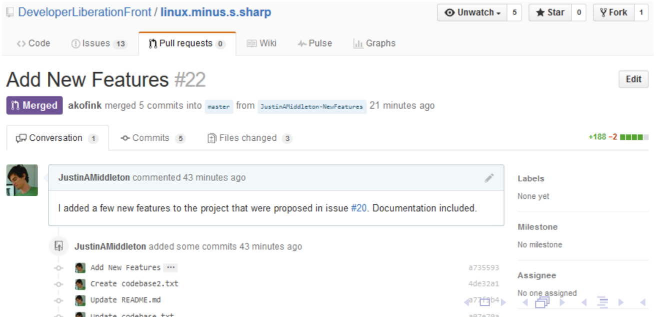
Forking a repo

- 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



Add New Features #22

Merged akofink merged 5 commits into master from JustinAMiddleton-NewFeatures 21 minutes ago

JustinAMiddleton commented 43 minutes ago

I added a few new features to the project that were proposed in issue #20. Documentation included.

JustinAMiddleton added some commits 43 minutes ago

```
Add New Features ···        a735593
Create codebase2.txt        4de32a1
Update README.md            a77f3b4
Update codebase.txt
```

Labels
None yet

Milestone
No milestone

Assignee
No one assigned

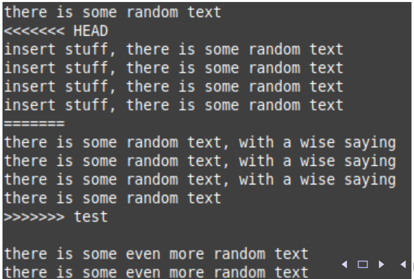
Pair Activity

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?

How merge conflicts occur

- when merging, git tries to figure out how to integrate changes
- sometimes, git needs human help, espeically, when two branches touch the same file
- git will mark problematic areas, you will not be able to commit once you resolve them
- you can use dedicated merge tools: e.g. DiffMerge



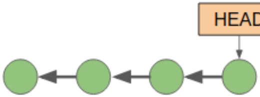
```
there is some random text
<<<<<<< HEAD
insert stuff, there is some random text
insert stuff, there is some random text
insert stuff, there is some random text
insert stuff, there is some random text
=======
there is some random text, with a wise saying
there is some random text, with a wise saying
there is some random text, with a wise saying
there is some random text
>>>>>>> test

there is some even more random text
there is some even more random text
```

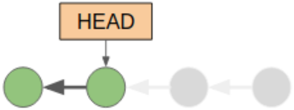
Resetting, Checking out and Reverting

- reset, checkout and revert allow you to undo some change to your repo
- reset: takes a specified commit and resets to match the state of repo at that specific commit
- checkout: moves the HEAD ref pointer to a specific commit
- revert: undo a commit by creating a new commit

Reset a specific commit

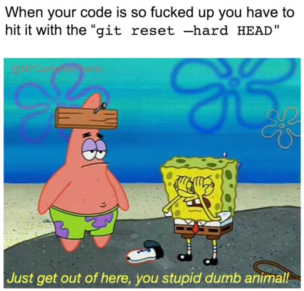
- resetting is a way to move the tip of branch to a different commit
- `git reset` is a simple way to undo changes that haven't been shared by others
- `git reset HEAD~2`
- oprhaned commits will be deleted next time git does a garbage collection





reset options

- `--soft`: undo commit and put files back onto stage
- `--mixed`: staged snapshot updated to match specified commit, but working directory not affected (default)
- `--hard`: undo the last commit, unstage files and undo and changes in the working directory


When your code is so fucked up you have to hit it with the "git reset –hard HEAD"
@NPCompleteTeens
Just get out of here, you stupid dumb animal!

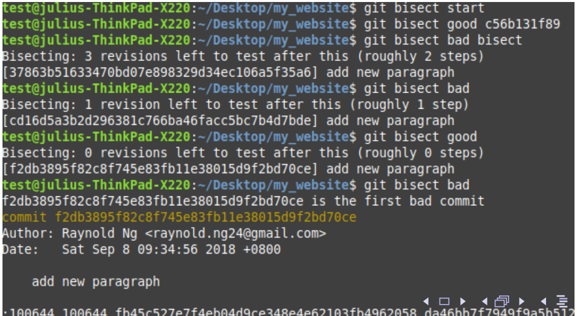
Checkout old commits

- `git checkout`: used to update the state of the repository to a specific point in the projects history
- useful for quickly inpsecting an old version of your project
- detached HEAD: no branch reference to HEAD, no way to access new commits if you commit them. So always create a new branch before adding commits

Nifty Tricks: bisect

- `git bisect` does a bianry search through your commit history to help you identify the commit that caused the issue

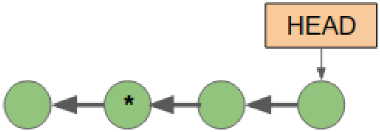
```
git bisect start
git bisect good <commit>
git bisect bad <commit>
```

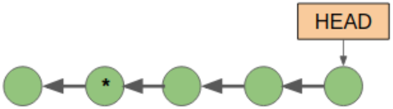


```
test@julius-ThinkPad-X220:~/Desktop/my_website$ git bisect start
test@julius-ThinkPad-X220:~/Desktop/my_website$ git bisect good c56b131f89
test@julius-ThinkPad-X220:~/Desktop/my_website$ git bisect bad bisect
Bisecting: 3 revisions left to test after this (roughly 2 steps)
[37863b51633470bd07e898329d34ec106a5f35a6] add new paragraph
test@julius-ThinkPad-X220:~/Desktop/my_website$ git bisect bad
Bisecting: 1 revision left to test after this (roughly 1 step)
[cd16d5a3b2d296381c766ba46facc5bc7b4d7bde] add new paragraph
test@julius-ThinkPad-X220:~/Desktop/my_website$ git bisect good
Bisecting: 0 revisions left to test after this (roughly 0 steps)
[f2db3895f82c8f745e83fb11e38015d9f2bd70ce] add new paragraph
test@julius-ThinkPad-X220:~/Desktop/my_website$ git bisect bad
f2db3895f82c8f745e83fb11e38015d9f2bd70ce is the first bad commit
commit f2db3895f82c8f745e83fb11e38015d9f2bd70ce
Author: Raynold Ng <raynold.ng24@gmail.com>
Date:   Sat Sep 8 09:34:56 2018 +0800

    add new paragraph

:100644 100644 fb45c527e7f4eb04d9ce348e4e62103fb4962058 da46bb7f7949f9a5b512
```

Undoing public commits with revert

- reverting undoes a commit by creating a new commit
- safe way to undo changes as it will not rewrite commit history
- `git revert HEAD~2`



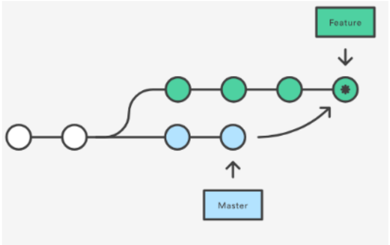


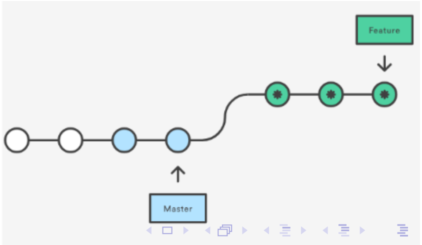
git checkout file

- checking out a file similar to `git reset` with a file path, except it updates working directory instead of the stage
- it does not move the move the HEAD reference, so you won't switch branches

Rebasing

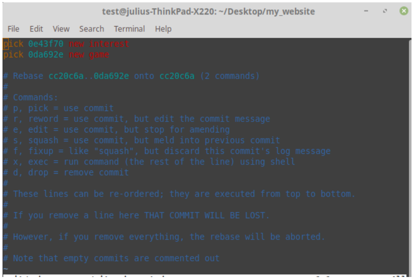
- rebase solves the smae problem as git merge, both are designed to integrate changes from one branch to another
- rebasing doesn't have the extraneous merge commit which can pollute your branch history if you are doing a lot of merges
- rebasing moves the entire branch to begin on the tip of master branch, incoporating all the new commits in master





Interactive Rebasing

```
git checkout feature
git rebase -i master
```


test@julius-ThinkPad-X220: ~/Desktop/my_website
File Edit View Search Terminal Help

```
pick 0e43f70 new interest
pick 0da692e new game

# Rebase cc20c6a..0da692e onto cc20c6a (2 commands)
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
# d, drop = remove commit
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
~
```

Where are we?

We have covered:

- setting up a repository
- saving changes
- undoing changes
- inspecting and rewriting history
- collaborating online

What is next?

Some things we didn't cover:

- Cherry picking
- git wrappers: sourcetree, magit, smartgit

Talk to us!

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