Commits and Commit Messages

What is a “commit”?

- Small set of modifications to a code base
  - Each commit should contain one (atomic) change
  - Commits should be standalone (independent of other commits)

Open Source Examples

- Chromium
  - https://github.com/chromium/chromium/commits/master
- NumPy
  - https://github.com/numpy/numpy/commits/master
- Evennia (python text-based game library)
  - https://github.com/evennia/evennia/commits/master

Commits

- Do not fold small changes (e.g. typo fixes) into another commit
- When commits are atomic and standlone, they can be applied and reverted independently
- The **commit message** summarizes the code changes
- Makes the version control utilities much easier to use

Commit Messages

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Guidelines

- Commit message guidelines differ by company.
- Bigger organizations programmatically read/modify commit messages.
  - Chromium's commit messages have a lot of boilerplating.

Common guidelines

From https://chris.beams.io/posts/git-commit/

- Separate subject from body with a blank line.
- Limit the subject line to 50 characters.
- Capitalize the subject line.
- Do not end the subject line with a period.
- Use the imperative mood in the subject line.
- Wrap the body at 72 characters.
- Use the body to explain what and why vs. how.

Example:

You add the following lines to a file called “tictactoe.py”

```
+ # Python TicTacToe game on the command line
+ # Author: Mr. Pirate <mr@pirate.com>
```

What should your commit message be?

Example:

What is wrong with the following commit messages?

```
Added comments to tictactoe
author and file description in tictactoe.py
comment description and author and email to the first few l
Add description, author, and email.
```

Your Project Commits

- Should be non-trivial (can't just be fixing a typo)
- Should be adding a major functionality
- Should have a commit message that follows the "Common guidelines"
- Should have code that follows the coding guidelines discussed today

Reviewing Code

Code Review

Most companies use "code review" to ensure high code quality.

1. Code writer submits code for review.
2. One or more reviewers (peers) read the code.
3. If reviewers notice areas of improvement, reviewers will request for changes.
4. Code writer works with the reviewer to address any raised issue (back to step 2)
5. When all reviewer concerns are addressed, the code is accepted (pushed).

Why code review?

- Encourage committers to write clean code.
- Share knowledge across team members.
- Encourages consistency in the code base.
- Help prevent bugs and other issues.

In most large organizations, **all** code, no matter who wrote it or how large/small it is, need to be reviewed.

What does the reviewer do?

- Does the code accomplish the author's purpose?
- What is the author's approach? Would you have solved the problem differently?
- Do you see potential for useful abstractions?
- Do you spot any bugs or issues?
- Does the change follow standard patterns?
- Is the code easy to read?
- Is this code documented and tested?

Example:

- https://github.com/evennia/evennia/pull/1666
- https://github.com/numpy/numpy/pull/11721
- https://github.com/numpy/numpy/pull/10931
- https://github.com/numpy/numpy/pull/10771

How to Review Code

- Critique the code, not the author.
  - "*your* code has a bug" vs "the code has a bug".
- Ask questions (perception checking!).
- Reviews should be concise and actionable:
  - Make it clear what you are asking for
- Don't be mean.

Responding to Code Review

- Be civil, and be open minded.
- First time you submit code, you will have *many* comments, don't feel daunted.

Setting Up Your Project Repository

Your repository should . . .

- Have meaningful directory structure
- Have a “README.md” that contains:
  - A description of the game
  - Instructions on how to install and run the game
  - Instructions on how to play the game
  - License information
  - Author information
- Have a clean commit history
- A “.gitignore” file that prevents machine-generated files from being committed

Examples

https://github.com/tasdikrahman/spaceShooter
https://github.com/stephank/arashi-js/blob/master/HACKING.md
https://github.com/mbostock/polly-b-gone/wiki

Clean commit history

- If you are not merging branches, use `git pull --rebase` instead of `git pull`
- Learn about `git rebase --amend`
- Learn about `git rebase -i`
- (Be careful to only rewrite history locally)