Crafting git commits

A well-crafted git commit message is the best way to communicate context about a change to fellow developers and your future self.

Example

```
Short (72 chars or less) summary

More detailed explanatory text. Wrap it to 72 characters. The blank
line separating the summary from the body is critical (unless you omit
the body entirely).
Write your commit message in the imperative: "Fix bug" and not "Fixed
bug" or "Fixes bug." This convention matches up with commit messages
generated by commands like git merge and git revert.

Further paragraphs come after blank lines.

* Bullet points are okay, too.

- Typically a hyphen or asterisk is used for the bullet, followed by a
single space.
```

The first line of your commit

- Succinctly explain the headline of the change.
- Written in the imperative: "Fix bug" and not "Fixed bug" or "Fixes bug."
- Appears in `git log` so it doesn't need to be the whole story.
- Doesn't need to explain the *why* - this is for the commit body.
- Keep it under 72 chars so it doesn't wrap.
- Consider prefixing it according to our conventional commit guidelines, e.g., `chore: Bump y18n` or `style: Use camelCase for Fooble`.
- Should start with a capital letter, e.g. "Fix typo" instead of "fix typo".

- Should not end in a period.

The commit body

- Put a blank line between the first line and the body.
- Describe *why* a change is being made.
- Don't assume the person reading the commit understands what the original problem was.
- In most cases, leave out details about how a change has been made (the code shows this).
- Link to Trello, Slack, Notion, GitHub issues, Stack Overflow answers, or other informative urls.
    - But don't assume Trello, Slack or a Github PR, etc. will live forever. Where you include a link, make sure you're recording some high-level information about what the link contains.
- If appropriate, mention alternative approaches considered.
- Try to give insight into your decision-making process where possible.
    - Did you have some questions that you had to work through in order to complete this work? Is it possible someone looking at this code later will have the same questions? The commit message is a great place to record your thought process for posterity.
    - This is *especially* true for bug-fixes—a detailed commit message is a wonderful resource to future maintainers to document how you approached debugging and identified the root cause that your committed code is fixing.

Rebase your branch

By squashing commits down into atomic changes, it's easier to revert changes or review changes as a whole when looking back to see why something changed. GitHub pull requests are great for discussing and reviewing changes and the work in progress commits make perfect sense in the context of the history of a PR, but they don't make sense in the context of the history of the main branch git log.

Remember that the discussion on a Github PR won't display in the git history or in `git blame`.

A few good guides on rebasing are:

- The Git Book
- Github rebasing guide
- Merging vs Rebasing

Further reading

- http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html
- https://robots.thoughtbot.com/5-useful-tips-for-a-better-commit-message
- https://tekin.co.uk/2019/02/a-talk-about-revision-histories (watching, not reading)