Git Collaboration

Tidy commit history

- Reviewing by commits
- Each commit should make sense on its own with a single purpose. E.g. a new a feature and a bug discovered when developing it should not be in the same commit.

```
git commit -m "bug fix: ..."
git commit -m "add support for ..."
```

- Use fixup commits to let the reviewer (and yourself) know what commits are you fixing.
- When development is finished, squash unnecessary commits before merging, will be easy and quick with `--autosquash` if using fixups.
- GitHub knows to display diffs even after overriding history, don't be afraid to `push -f` (when amending or rebasing)

Commit message

- Should explain (Depending on the contents of the Jira):
    i. What we changed
    ii. Why we changed
    iii. How does the change solve the problem
    iv. For non trivial code, add links to sources
- By reading the commit message, the reviewer should know exactly what he's going into, without reading a single line of code
- Something unclear in the code? explain it in the commit message. This "comment" is always true to the commit and will forever remain
- No one likes to read code. if you can summarize your 300 lines change into 3 sentences, a year from now, I would prefer to read a bit of English and not the code

For easy tracking

- In commit message: link(s) to Jira if exists
- In Jira, make sure the PR(s) are linked (via GitHub plugin or simply link in the comment)
- Optional: PR links across different repos for the same feature

Reviewing

The developer:

- Review your PR before submitting for others to review.

The reviewer:

- If you think two changes should not be in the same commit, request to separate them.
- If you think the commit message didn't explain well the code you reviewed, request to explain better.