Version Control Guidelines

Commit Guidelines

Commit Related Changes

- A commit should be a wrapper for related changes.
- Small commits make tit easier for reviewers and future readers to understand the changes
- With tools like the staging area and the ability to stage only parts of a file, Git makes it easy to create very granular commits.

  Fixing 2 different bugs should produce 2 separate commits

  Refactoring-changes and functional-changes should never be part of the same commit.

- If you need to commit trivial changes, such as renaming a variable created in the previous commit, or fixing a typo, consider doing

  > git commit –amend

Commit Often

- Committing often keeps your commits small and, again, helps you commit only related changes.
- Moreover, it allows you to share your code more frequently with others.
- That way its easier for everyone to integrate changes regularly and avoid having merge conflicts.
- Having few large commits and sharing them rarely, in contrast makes it hard both to solve conflicts and to comprehend what happened.

Don't Commit Half Done Work

- Only commit complete changes, but split large features into small, logical chunks and remember to commit early and often.
- If you are tempted to commit just because you need a clean working copy (to check out a branch, pull in changes), consider using Git's Stash feature instead

Test Before Commit

- Resist the temptation to commit something that you "think" is completed.
- Test it thoroughly to make sure it really is completed and has no side effects (as far as one can tell)
- While committing half-baked things in your local repository only requires you to forgive yourself, having your code tested is even more important when it comes to pushing or sharing your code with others.

Write Good Commit Messages

- Begin your message with a short summary of your changes.
- Separate it from the following body with a blank line.
- The body of your message should provide detailed answers to the 3 questions:
    - Why is this change necessary?
    - How does this change address the issue?
    - What side effects does this change have?
- Use the imperative, present tense ("change", not "changed" or "changes") to be consistent with generated messages from commands like git merge.
- Think of your commit message as an email you are sending to all the developers who are going to try to understand your changes in the future, possibly yourself.

Use Branches

- Branching is one of Git's most powerful features – and this is not by accident: quick and easy branching was a central requirement from day one.
- Branches are the perfect tool to help you avoid mixing up different lines of development.

- You should use branches extensively in your development workflows: for new features, bug fixes, experiments, ideas etc.

Pull Request Guidelines

Useful Commit Messages & PR Description

- Just remember these 3 questions,
  - Why is this change necessary?
  - How does this change address the issue?
  - What side effects does this change have?

Small and Contained

- Add only code changes that are strictly related to the issue
- Use separate commits to distinguish the steps taken to implement the change
- Make pull requests as small as possible: easier and faster to review
- Never mix refactoring changes and functional changes in the same commit

Guided

- Add annotations to guide the reviewer through your changes
- Clarify the requirements (details reasons and context for this change)
- Add objectives for the reviewers:
  - Show which files to look at first
  - Point out the important changes
- Defend the reason and methods for each code modification