Git Good Practices

If you're reading this document, you're probably looking for some guidance on how to use Git on your project. There are a lot of ways to use it, both good and bad. So, we want to help you wade through the different methods and help answer the question: "How do we actually do it here at Fearless"?

This document outlines the foundational ways we expect everyone at Fearless to use Git. However, we recognize that not every project will be able to use Git in the way we're asking. That's why these are "good practices" and not "best practices" — these are methods we've tested that will produce good outcomes, but they aren't strict standards. Use them when you can and use your best judgment when you can't!

At the end of this doc, you'll also find a list of additional materials — resources, repos, tools, and playgrounds — where you can learn more about Git and play around with it a bit. We encourage you to poke around those links and let us know if anything should be added to the list!

Our Git good practices

Pull requests

Pull requests should be reviewed in a timely manner. Once the branch is merged, it should be deleted.

If changes are made to a pull request, the original reviews should be dismissed after it has been reviewed.

Team members are highly encouraged to share WIP branches and create draft pull requests. A draft pull request should not be marked as "Ready to Review" until indicated work has been completed and the branch is ready for code review. Pull requests live in the repository forever.

A good pull request should include the following:

- One self-contained change
- Ideally, no more than 250 lines of change
- Self-explanatory title that describes what the pull request does
- Description that details what was changed, why it was changed, and how it was changed
- Link to ticket
- Links to any related pull requests

8 Market Place, Suite 200, Baltimore, MD 21202
(410) 394-9600 / fax (410) 779-3706 / fearless.tech

gitignore

A project repository's gitignore file should be project specific. It **should not** include IDE- and OS-specific file information.

Every developer should have a global gitignore file for their system. Because IDE- and OS-specific files are specific to the developer environment, they **should** be added to a global gitignore.

Smaller commits

Developers should be very intentional about their commits and avoid `git add *` `git add .` These commands have the possibility of bringing in unintended changes.

Each commit should represent a finite change. If many different changes are made at once, a developer should leverage line-specific commits to maintain the integrity of the repository. Commits done this way make it easier for other developers to see and review the changes in context. They also create less potential for merge conflicts.

Formatting changes and logic changes should be separate commits for the purpose of maintaining code. It's easier to identify a bug in a finite logic change than to have to parse a large mixed-purpose commit.

Commit messages should be clear about the change and why it was made. A well-written commit message helps with future maintainability of the code base.

Git force-with-lease

`Git push -- force`forcibly overwrites the remote commit history with your own local history and **should never be used without discussion with your team.**

If a regular push command is not effective, the developer should first pull any commits from the target branch that do not exist in the local branch with **`git fetch`** before using the force command as it is normally reserved for extreme cases when something has gone wrong with the repository.

If a force push is necessary, developers should use **`git push –force-with-lease`**, which will only overwrite the remote branch if your local history is aware of all commits on the remote branch. Using **`git push –force-with-lease`** safeguards the developer from destroying someone else's changes to the codebase.

8 Market Place, Suite 200, Baltimore, MD 21202
(410) 394-9600 / fax (410) 779-3706 / fearless.tech

Scenario	git push	git push --force-with-lease
New commits on local branch	works	works
New commit on remote branch added by another user	FAILS	FAILS
Commit from remote branch is modified on local branch	FAILS	works

Debugging

Git can and should be used for debugging.

`Git blame` is useful for identifying and understanding a specific change; the commit message should provide enough valuable information for another developer to pick up where the original author left off. If the commit message is unclear, the developer can also ask the author for clarification.

Branch structures

Preferred: Trunk Based Development

In Trunk Based Development, developers collaborate in a single branch called the trunk, typically named “main”. Following the pull request workflow, short lived feature branches are code reviewed and merged before integrating into the trunk.

Branches

1. Main
2. Feature-X

8 Market Place, Suite 200, Baltimore, MD 21202
(410) 394-9600 / fax (410) 779-3706 / fearless.tech

Process

- Main is releasable anytime
- Main is tagged to deploy to production
- Devs must branch off Main branch
- Pull requests for Feature-X branches are submitted and reviewed daily
- Feature-X branches are merged into Main branch

Rationale

- Because Main is releasable anytime, the team has the agility to frequently deploy to production
- Requirement for continuous integration and continuous delivery
- Ensures teams release code quickly and consistently
- Code reviews are more efficient; small branches mean engineers can quickly see and review changes
- Limits long-lived branches, reducing the likelihood of merge conflicts and cognitive overload of large amounts of changes

Potential pitfalls

Specifically with GitHub, it's not possible to "protect" a tag. So anyone with maintainer access is allowed to tag and could trigger a push to production. Though some CI tools, such as CircleCI, could have further levels of protection.

Feature flags are required to manage releases between production and non-production environments.

Best suited for

- A mature engineering team
- A loosely coupled code base that can support feature abstractions

Next best: Git Flow

Sometimes teams or clients need an intermediate step between where they are and where they need to go in order to build the necessary comfort and positivity. In those cases, Git Flow may be the right choice.

Branches

1. Main

8 Market Place, Suite 200, Baltimore, MD 21202
(410) 394-9600 / fax (410) 779-3706 / fearless.tech

2. Dev
3. Feature-X

Process

- Main is releasable anytime
- Main is tagged to deploy to production
- Developers must branch off Dev branch
- Dev is merged into Main on a “regular” basis
- Dev is rebased on Main when hotfixes are committed

Rationale

A Dev branch creates a perception of safety for the developers who are concerned about committing to main and possibly affecting deploys.

Tags on main are used for deployment to give more control over when deployments to production occur.

Potential pitfalls

Merging Dev into Main consistently can be time consuming. One also must take care to rebase Dev on Main when hot fixes are produced, and this can create ripple effects on feature branches. Main and Dev branch split is redundant and impedes the establishment of continuous integration and continuous delivery.

Best suited for

- Less experienced engineering team
- Open source projects
- Large projects compiling releases
- Projects that have scheduled release cycles

Tips & Tricks

Git Bisect

In the scenario where a team does not know when a bug was introduced, they can leverage `git bisect` to do a binary search of the repository to narrow down the issue to a specific commit.

8 Market Place, Suite 200, Baltimore, MD 21202
(410) 394-9600 / fax (410) 779-3706 / fearless.tech

Additional materials

Fearless repositories

- https://github.com/FearlessSolutions
- https://github.com/orgs/FearlessFarms

Resources

- Git documentation
    - Git documentation straight from the source
- Basic Git commands
- Create a global gitignore
    - Instructions for setting up a global gitignore file
- Project gitignore templates
- Write better commit messages
- Writing a great pull request description
- Benefits of making small pull requests
- Merge vs Rebase
- Trunk based development
- Pre-commit hooks
- Never `git push force`

Playgrounds

- Interactive git branching demo
    - Interactive git branching - no demo
- https://git-school.github.io/visualizing-git/

Tools

- Command line
- GitHub Desktop
- GitX
- Trufflehog
    - Pre-commit hook to catch committed secrets

8 Market Place, Suite 200, Baltimore, MD 21202
(410) 394-9600 / fax (410) 779-3706 / fearless.tech