DevOps

Week 04

Murtaza Munawar Fazal

Git Branch Workflow

- When evaluating a workflow for your team, you must consider your team's culture. You want the workflow to enhance your team's effectiveness and not be a burden that limits productivity. Some things to consider when evaluating a Git workflow are:
  - Does this workflow scale with team size?
  - Is it easy to undo mistakes and errors with this workflow?
  - Does this workflow impose any new unnecessary cognitive overhead on the team?

Common Branch Workflow

- Most popular Git workflows will have some sort of centralized repo that individual developers will push and pull from.
- These comprehensive workflows offer more specialized patterns about managing branches for feature development, hotfixes, and eventual release.
- Following are two popular Git workflows.
  - Trunk-based development
  - Forking workflow

Trunk Based Development

- Trunk-based development is a logical extension of Centralized Workflow.
- The core idea behind the Feature Branch Workflow is that all feature development should take place in a dedicated branch instead of the main branch.
- This encapsulation makes it easy for multiple developers to work on a particular feature without disturbing the main codebase.
- It also means the main branch should never contain broken code, which is a huge advantage for continuous integration environments.

Trunk Based Development

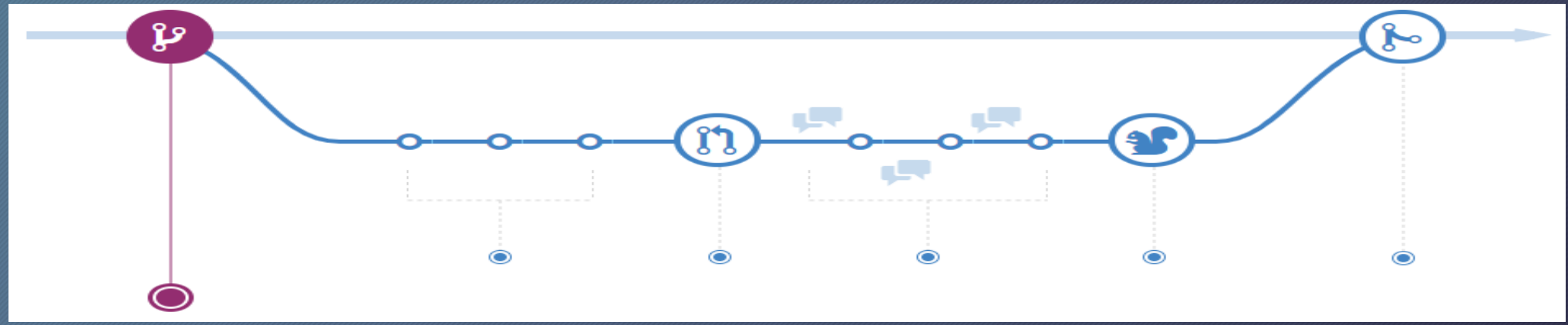
- Pull requests make it incredibly easy for your team to comment on each other's work. Also, feature branches can (and should) be pushed to the central repository. It allows sharing a feature with other developers without touching any official code.
- Since the main is the only "special" branch, storing several feature branches on the central repository doesn't pose any problems. It's also a convenient way to back up everybody's local commits.
- Feature branches should have descriptive names, like new-banner-images or bug-91. The idea is to give each branch a clear, highly focused purpose.
- Git makes no technical distinction between the main and feature branches, so developers can edit, stage, and commit changes to a feature branch.

Forking Workflow

- The Forking Workflow is fundamentally different than Trunk based development.
- Instead of using a single server-side repository to act as the "central" codebase, it gives every developer a server-side repository.
- It means that each contributor has two Git repositories:
  - A private local one.
  - A public server-side one.

Create a Branch

- When you're working on a project, you will have many different features or ideas in progress at any given time – some of which are ready to go and others that aren’t. Branching exists to help you manage this workflow.
- When you create a branch in your project, you're creating an environment where you can try out new ideas.
- Changes you make on a branch don't affect the main branch, so you're free to experiment and commit changes, safe in the knowledge that your branch won't be merged until it's ready to be reviewed by someone you're collaborating with.

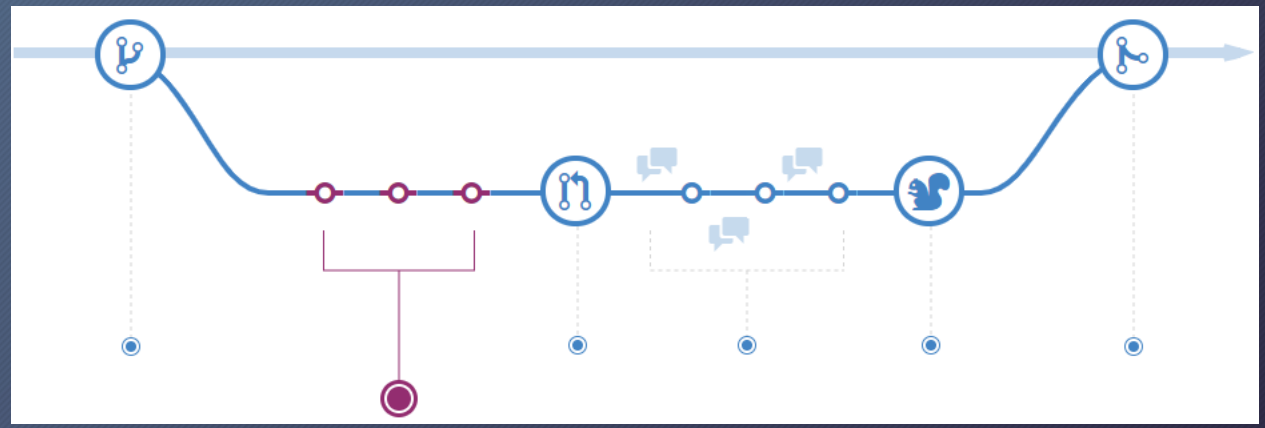


Create a Branch

- Branching is a core concept in Git, and the entire branch flow is based upon it. There's only one rule: anything in the main branch is always deployable.
- Because of this, your new branch must be created off the main when working on a feature or a fix.
- Your branch name should be descriptive (for example, refactor-authentication, user-content-cache-key, make-retina-avatars) so that others can see what is being worked on.

Add Commits

- Once your branch has been created, it's time to start making changes. Whenever you add, edit, or delete a file, you're making a commit and adding them to your branch. This process of adding commits keeps track of your progress as you work on a feature branch.
- Commits also create a transparent history of your work that others can follow to understand what you've done and why.
- Each commit has an associated commit message, which explains why a particular change was made.
- Furthermore, each commit is considered a separate unit of change. It lets you roll back changes if a bug is found or you decide to head in a different direction.
- Commit messages are essential, especially since Git tracks your changes and then displays them as commits once pushed to the server.
- By writing clear commit messages, you can make it easier for other people to follow along and provide feedback.



Open a Pull Request (PR)

- The Pull Requests start a discussion about your commits. Because they're tightly integrated with the underlying Git repository, anyone can see exactly what changes would be merged if they accept your request.
- You can open a Pull Request at any point during the development process when:
  - You've little or no code but want to share some screenshots or general ideas.
  - You're stuck and need help or advice.
  - You're ready for someone to review your work.
- In your Pull Request message, you can ask for feedback from specific people or teams, whether they're down the hall or 10 time zones away.
- Pull Requests help contribute to projects and for managing changes to shared repositories.
- If you're using a Fork & Pull Model, Pull Requests provide a way to notify project maintainers about the changes you'd like them to consider.
- If you're using a Shared Repository Model, Pull Requests help start code review and conversation about proposed changes before they're merged into the main branch.

Merge

- Once your changes have been verified, it's time to merge your code into the main branch.
- Once merged, Pull Requests preserve a record of the historical changes to your code. Because they're searchable, they let anyone go back in time to understand why and how a decision was made.
- By incorporating specific keywords into the text of your Pull Request, you can associate issues with code. When your Pull Request is merged, the related issues can also close.
- This workflow helps organize and track branches focused on business domain feature sets.

Collaborate with Pull Request

- Pull requests let you tell others about changes you've pushed to a GitHub repository.
- Once a pull request is sent, interested parties can review the set of changes, discuss potential modifications, and even push follow-up commits if necessary.
- Pull requests are commonly used by teams and organizations collaborating using the Shared Repository Model.
- Many open-source projects on GitHub use pull requests to manage changes from contributors.
- They help provide a way to notify project maintainers about changes one has made.

Collaborate with Pull Request

- Also, start code review and general discussion about a set of changes before being merged into the main branch.
- Pull requests combine the review and merge of your code into a single collaborative process.
- Once you're done fixing a bug or new feature in a branch, create a new pull request.
- Add the team members to the pull request so they can review and vote on your changes.
- Use pull requests to review works in progress and get early feedback on changes.
- There's no commitment to merge the changes as the owner can abandon the pull request at any time.