What is the best Git branch strategy?

Git and other version control systems give software developers the power to track, manage, and organize their code.

In particular, Git helps developers collaborate on code with teammates; combining powerful features like commits and branches with specific principles and strategies helps teams organize code and reduce the time needed to manage versioning.

Of course, every developer and development team is different, with unique needs. Here is where a Git branching strategy comes in.

We will be covering three fairly popular Git branch strategies, each with their own benefits. The best part? None of these workflows are set in stone and can, and should, be modified to fit your specific environment and needs.

Please note: many of these original strategies refer to 'master' branches, but we have chosen to use 'main' instead.

No matter which branching strategy you choose, GitKraken enables powerful, easier, and safer collaboration with Git with features like predictive merge conflict detection and in-app pull requests.

- Main
- Develop
- Feature
- Release
- Hotfix

The two primary branches in Git flow are *main* and *develop*. There are three types of supporting branches with different intended purposes: *feature*, *release*, and *hotfix*.

Git Flow: Pros & Cons

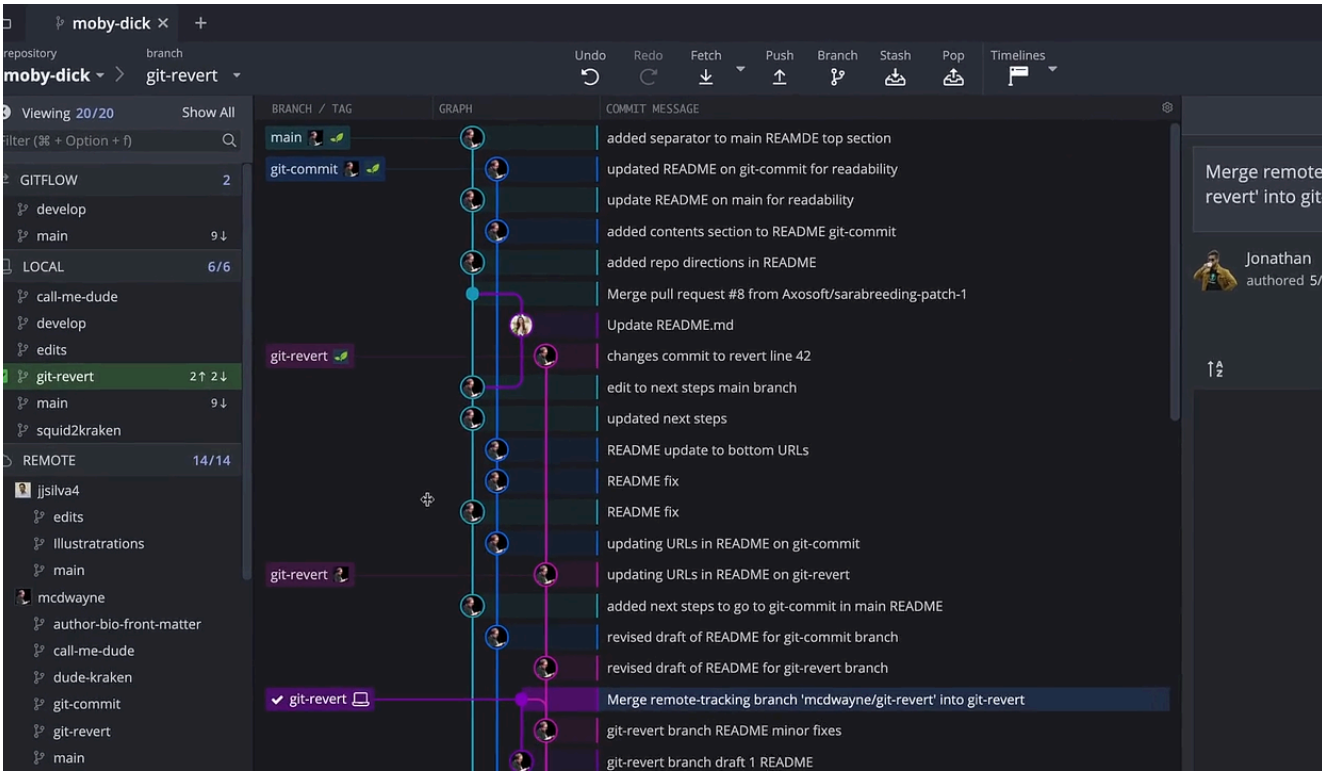
The Git flow branching strategy comes with many benefits, but does introduce a few challenges.

The Benefits of Git Flow:

1. The various types of branches make it easy and intuitive to organize your work.
2. The systematic development process allows for efficient testing.
3. The use of release branches allows you to easily and continuously support multiple versions of production code.

The Challenges of Git Flow:

1. Depending on the complexity of the product, the Git flow model could overcomplicate and slow the development process and release cycle.
2. Because of the long development cycle, Git flow is historically not able to support Continuous Delivery or Continuous Integration.



To initialize Git flow with GitKraken, open your repo and then navigate to `Preferences` → `Gitflow` to set your preferred branch naming conventions. GitKraken will then help you start and finish feature, release, and hotfix branches.

GitKraken offers incredible GitHub integrations (https://www.gitkraken.com/integrations/github), GitLab integrations (https://www.gitkraken.com/integrations/gitlab), Bitbucket integrations (https://www.gitkraken.com/integrations/bitbucket), and Azure DevOps integrations (https://www.gitkraken.com/integrations/azure-devops) to make it easy to work with hosted repositories.

GitKraken empowers teams large and small to harness the true power of Git, giving you more visibility into who is working on what and when, so you can avoid conflicts and secure your code.

GitHub Flow Branch Strategy

The GitHub flow branching strategy is a relatively simple workflow that allows smaller teams, or web applications/products that don't require supporting multiple versions, to expedite their work.

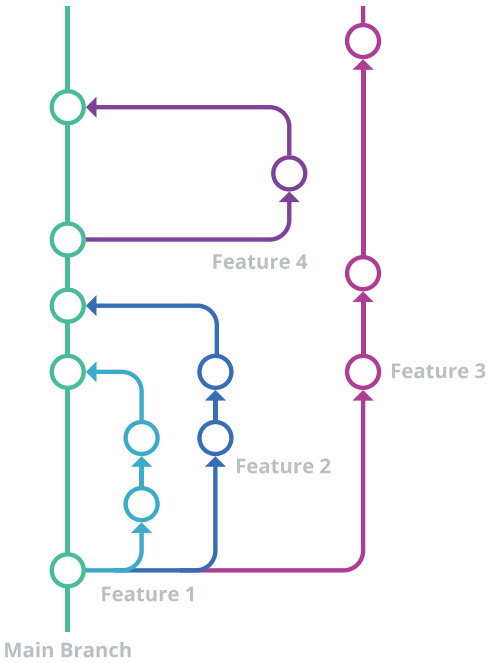
In GitHub flow (https://githubflow.github.io/), the main branch contains your production-ready code.

The other branches, feature branches, should contain work on new features and bug fixes and will be merged back into the main branch when the work is finished and properly reviewed.

GitHub Flow Considerations

While working with the GitHub flow branching strategy, there are six principles (https://githubflow.github.io/) you should adhere to to ensure you maintain good code.

1. Any code in the main branch should be deployable.
2. Create new descriptively-named branches off the main branch for new work, such as `feature/add-new-payment-types`.
3. Commit new work to your local branches and regularly push work to the remote.
4. To request feedback or help, or when you think your work is ready to merge into the main branch, open a pull request (https://www.gitkraken.com/learn/git/tutorials/what-is-a-pull-request-in-git).
5. After your work or feature has been reviewed and approved, it can be merged into the main branch.
6. Once your work has been merged into the main branch, it should be deployed immediately.



Custom image inspired by the GitHub Flow Guide. (https://guides.github.com/introduction/flow/)

The Benefits of GitHub Flow

1. Of the three Git branch strategies we cover in this post, GitHub flow is the most simple.
2. Because of the simplicity of the workflow, this Git branching strategy allows for Continuous Delivery and Continuous Integration.
3. This Git branch strategy works great for small teams and web applications.

The Challenges of GitHub Flow

1. This Git branch strategy is unable to support multiple versions of code in production at the same time.
2. The lack of dedicated development branches makes GitHub flow more susceptible to bugs in production.

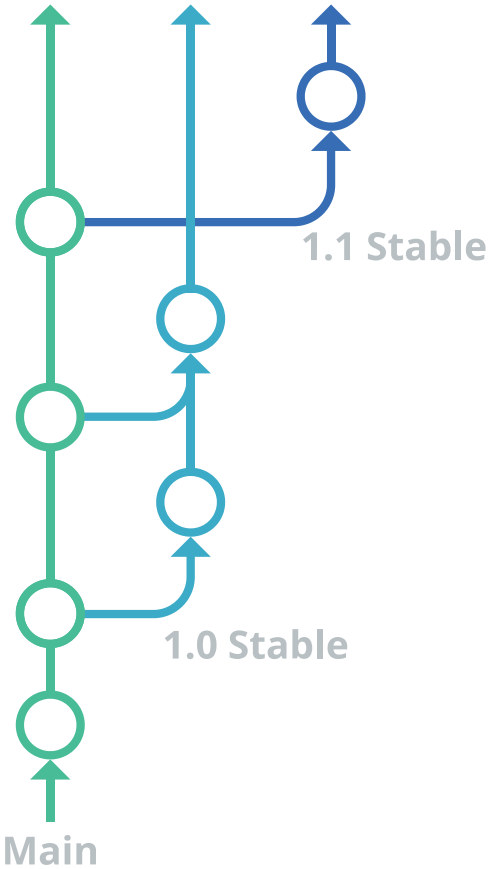
Just as in the other two Git branch strategies, GitLab flow (https://docs.gitlab.com/ee/topics/gitlab_flow.html) has a main branch that contains code that is ready to be deployed. However, this code is not the source of truth for releases.

In GitLab flow, the feature branch contains work for new features and bug fixes which will be merged back into the main branch when they're finished, reviewed, and approved.

Using GitLab flow in your release cycle

The GitLab flow branching strategy works with two different types of release cycles:

1. Versioned Release: each release has an associated release branch that is based off the main branch. Bug fixes should be merged into the main branch first, before being cherry-picked into the release branch.



Custom image inspired by image in GitLab Flow (https://about.gitlab.com/blog/2014/09/29/gitlab-flow/).

1. When compared to the Git flow branch strategy, GitLab flow is more simple.
2. GitLab flow is more organized and structured than the GitHub flow branch strategy.
3. After slight modification, GitLab flow can allow for Continuous Delivery and versioned releases.

Challenges of GitLab Flow

1. GitLab flow is not the simplest Git branch strategy.
2. GitLab flow is not the most structured Git branching strategy which can lead to messy collaboration.

The ability to visualize your commits and branches will give you and your team a newfound understanding of your repository's structure, making daily Git actions more intuitive.

Make Git Easier with GitKraken (/git-client/easy-git-features)

So, which is the best Git branching strategy?

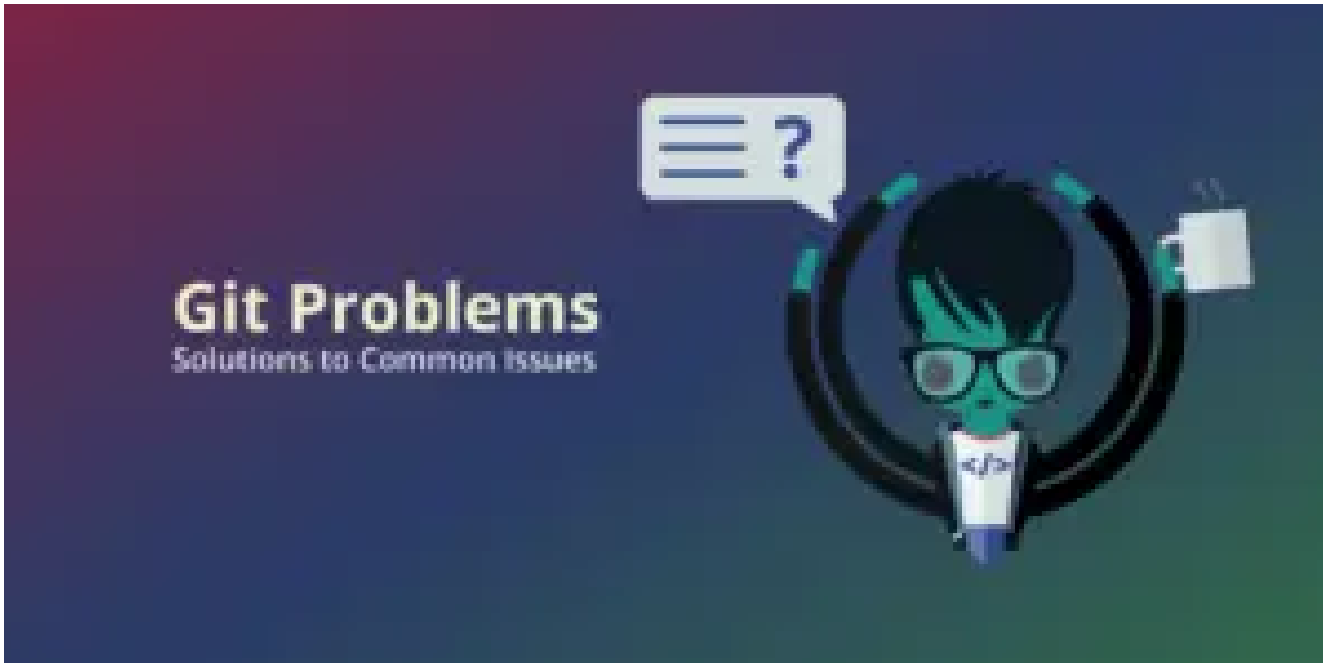
Ultimately, the answer to which Git branch strategy is the best depends on you and your team's environment, product and your specific development needs.

There is not a one-size-fits-all Git branch strategy, and regardless of which you end up selecting, it's likely you can optimize it with further modifications.

Get started with Git flow using GitKraken. Download the GitKraken Git GUI (https://www.gitkraken.com/git-client) for free.

(https://www.gitkraken.com/learn/git/git-fetch)

Git Fetch (https://www.gitkraken.com/learn/git/git-fetch)

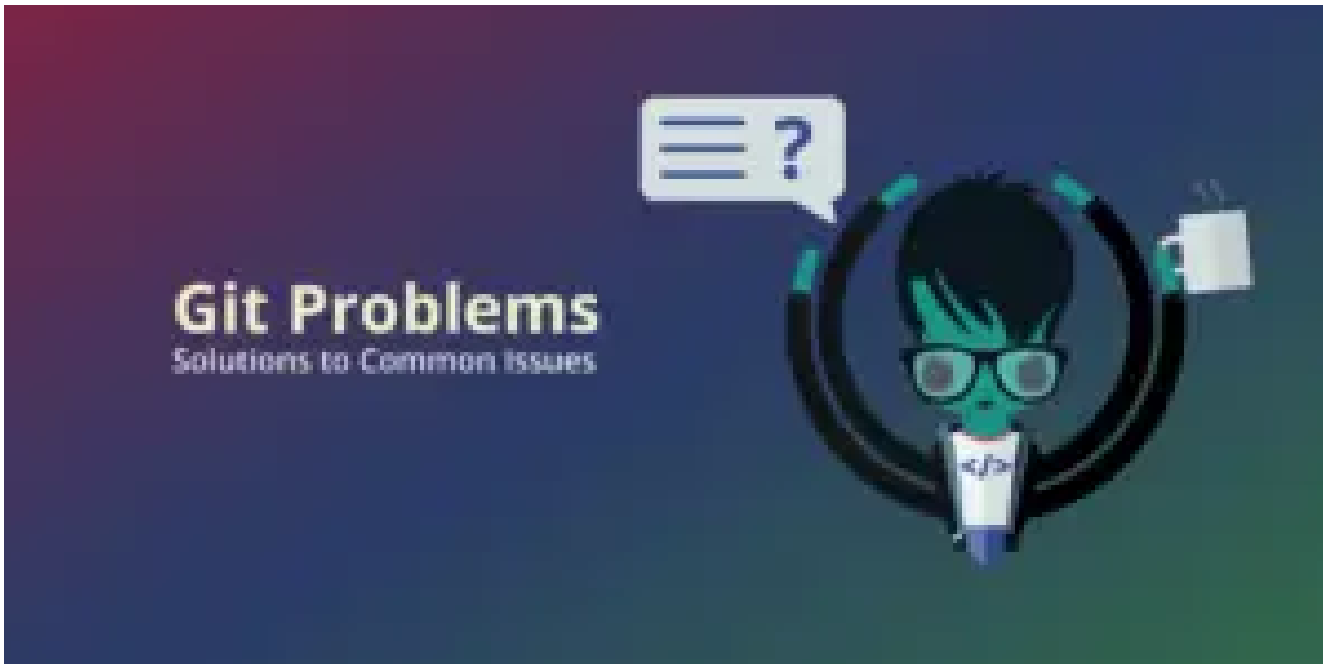


(https://www.gitkraken.com/learn/git/problems/git-push-to-remote-branch)

Git Push to Remote Branch (https://www.gitkraken.com/learn/git/problems/git-push-to-remote-branch)

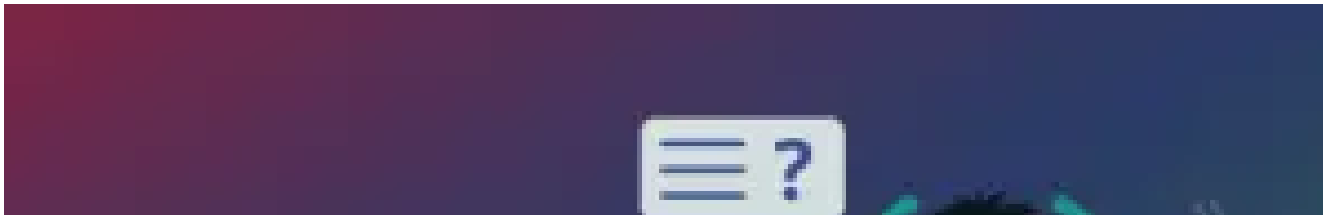
(https://www.gitkraken.com/learn/git/commands)

Commands (https://www.gitkraken.com/learn/git/commands)



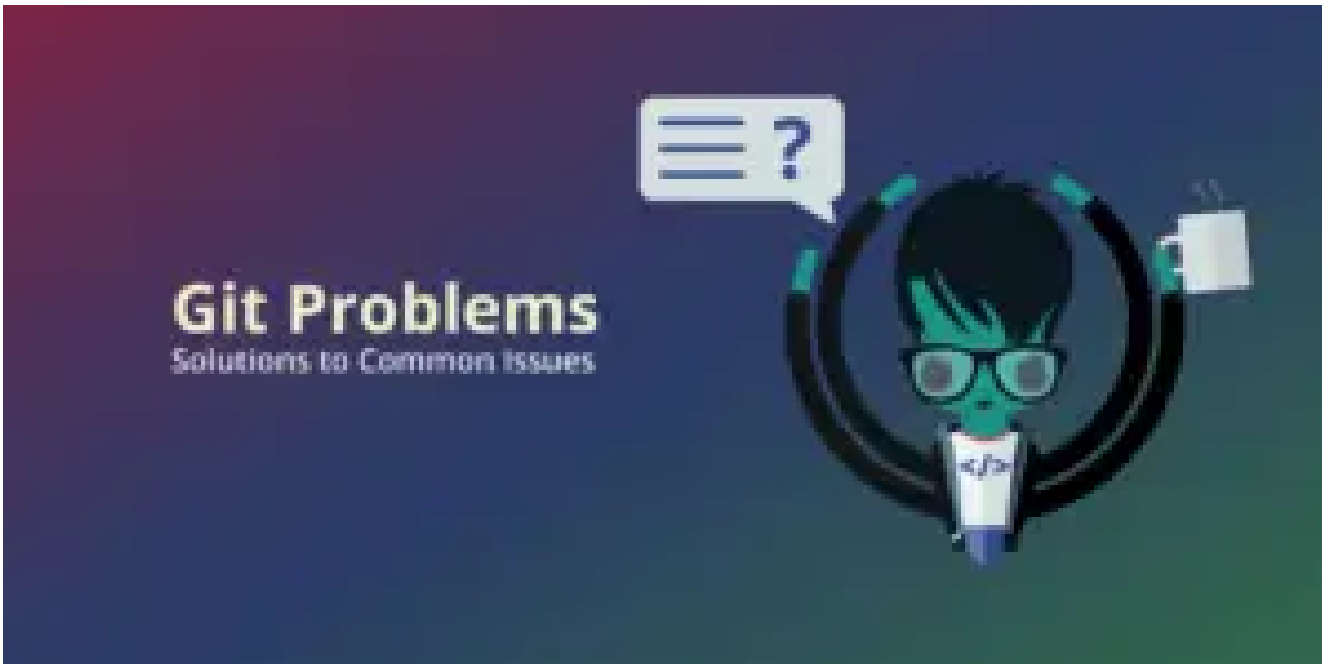
(https://www.gitkraken.com/learn/git/problems/switch-git-branch)

How do you switch a Git branch? (https://www.gitkraken.com/learn/git/problems/switch-git-branch)



(https://www.gitkraken.com/learn/git/problems/rename-git-branch)

How do you rename a Git branch? (https://www.gitkraken.com/learn/git/problems/rename-git-branch)



(https://www.gitkraken.com/learn/git/problems/pull-remote-git-branch)

Git Pull Remote Branch (https://www.gitkraken.com/learn/git/problems/pull-remote-git-branch)

PRODUCTS

Pricing (/pricing)

Roadmap (/git-client/roadmap)

Cheat Sheet (https://www.gitkraken.com/pdfs/gitkraken-git-gui-cheat-sheet)

CLI Cheat Sheet (/pdfs/gitkraken-cli-cheat-sheet)

GitLens (/gitlens)

Documentation (https://help.gitkraken.com/gitlens/gitlens-home/)

Git Integration for Jira (/git-integration-for-jira)

Cloud Docs (https://help.gitkraken.com/git-integration-for-jira-cloud/git-integration-for-jira-home-gij-cloud/)

Data Center/Server Docs (https://help.gitkraken.com/git-integration-for-jira-data-center/git-integration-for-jira-home-self-manged/)

Security & Trust (/git-integration-for-jira/security-and-trust)

COMMUNITY

Learn Git Library (/learn/git)

Git Commands Cheat Sheet (/pdfs/git-basics-cheat-sheet)

Git Blog (/blog)

GitKraken Labs (https://www.gitkraken.com/labs)

Git Conference (http://gitkon.com/)

Ambassador Program (/ambassador)

Newsletter (/newsletter)

Slack Community (https://www.gitkraken.com/join-slack-community)

GitKraken for Students (/github-student-developer-pack-bundle)

Store (/store)

Keif Gallery (/keif-gallery)

COMPANY

Contact Us (/contact)

About Us (/about)

Careers (/careers)

Customers (/sample-customers)

Media (/media)

GitKraken

s://twitter.com/gitkraken) ken.com/join-slack-community) ww.youtube.com/gitkraken) nkedin.com/company/gitkraken) s://facebook.com/gitkraken)