Gitflow workflow

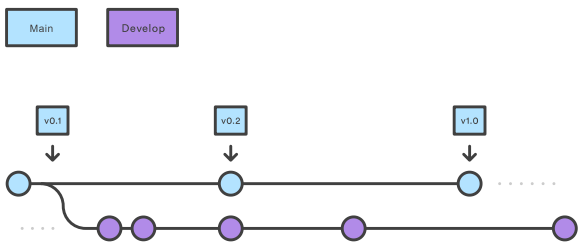
Gitflow is a legacy Git workflow that was originally a disruptive and novel strategy for managing Git branches. Gitflow has fallen in popularity in favor of trunk-based workflows, which are now considered best practices for modern continuous software development and DevOps practices. Gitflow also can be challenging to use with CI/CD. This post details Gitflow for historical purposes.

What is Gitflow?

Gitflow is an alternative Git branching model that involves the use of feature branches and multiple primary branches. It was first published and made popular by Vincent Driessen at nvie. Compared to trunk-based development, Gitflow has numerous, longer-lived branches and larger commits. Under this model, developers create a feature branch and delay merging it to the main trunk branch until the feature is complete. These long-lived feature branches require more collaboration to merge and have a higher risk of deviating from the trunk branch. They can also introduce conflicting updates.

Gitflow can be used for projects that have a scheduled release cycle and for the DevOps best practice of continuous delivery. This workflow doesn't add any new concepts or commands beyond what's required for the Feature Branch Workflow. Instead, it assigns very specific roles to different branches and defines how and when they should interact. In addition to `feature` branches, it uses individual branches for preparing, maintaining, and recording releases. Of course, you also get to leverage all the benefits of the Feature Branch Workflow: pull requests, isolated experiments, and more efficient collaboration.

How it works



Develop and main branches

Instead of a single `main` branch, this workflow uses two branches to record the history of the project. The `main` branch stores the official release history, and the `develop` branch serves as an integration branch for features. It's also convenient to tag all commits in the `main` branch with a version number.

The first step is to complement the default `main` with a `develop` branch. A simple way to do this is for one developer to create an empty `develop` branch locally and push it to the server:

```
git branch develop
```

version. Other developers should now clone the central repository and create a tracking branch for `develop`.

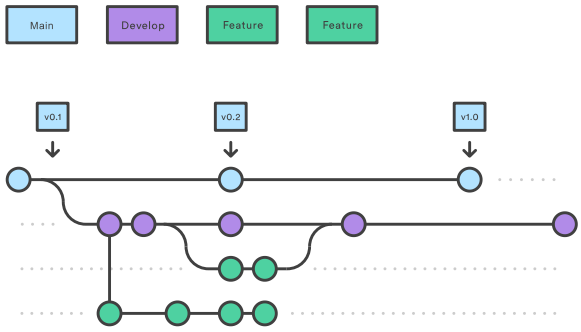
When using the git-flow extension library, executing `git flow init` on an existing repo will create the `develop` branch:

```
$ git flow init


Initialized empty Git repository in ~/project/.git/
No branches exist yet. Base branches must be created now.
Branch name for production releases: [main]
Branch name for "next release" development: [develop]


How to name your supporting branch prefixes?
Feature branches? [feature/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []


$ git branch
* develop
  main
```

Feature branches

Step 1. Create the repository

Each new feature should reside in its own branch, which can be pushed to the central repository for backup/collaboration. But, instead of branching off of `main`, `feature` branches use `develop` as their parent branch. When a feature is complete, it gets merged back into develop. Features should never interact directly with `main`.



Note that `feature` branches combined with the `develop` branch is, for all intents and purposes, the Feature Branch Workflow. But, the Gitflow workflow doesn't stop there.

`Feature` branches are generally created off to the latest `develop` branch.

Creating a feature branch

Without the git-flow extensions:

```
git checkout develop
git checkout -b feature_branch
```

```
git flow feature start feature_branch
```

Continue your work and use Git like you normally would.

Finishing a feature branch

When you're done with the development work on the feature, the next step is to merge the `feature_branch` into `develop`.

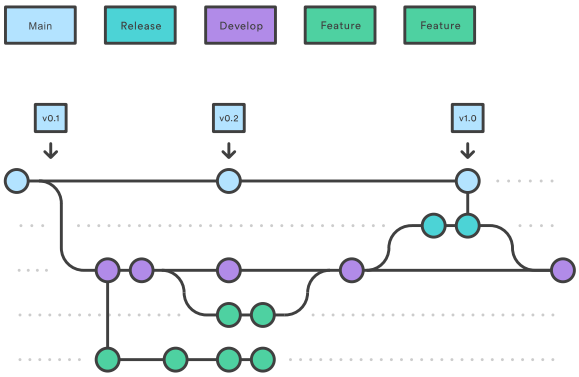
Without the git-flow extensions:

```
git checkout develop
git merge feature_branch
```

Using the git-flow extensions:

```
git flow feature finish feature_branch
```

Release branches



Once `develop` has acquired enough features for a release (or a predetermined release date is approaching), you fork a `release` branch off of `develop`. Creating this branch starts the next release cycle, so no new features can be added after this point—only bug fixes, documentation generation, and other release-oriented tasks should go in this branch. Once it's ready to ship, the `release` branch gets merged into `main` and tagged with a version number. In addition, it should be merged back into `develop`, which may have progressed since the release was initiated.

Using a dedicated branch to prepare releases makes it possible for one team to polish the current release while another team continues working on features for the next release. It also creates well-defined phases of development (e.g., it's easy to say, "This week we're preparing for version 4.0," and to actually see it in the structure of the repository).

Making `release` branches is another straightforward branching operation. Like `feature` branches, `release` branches are based on the `develop` branch. A new `release` branch can be created using the following methods.

Without the git-flow extensions:

```
git checkout develop
git checkout -b release/0.1.0
```

When using the git-flow extensions:

Once the release is ready to ship, it will get merged it into `main` and `develop`, then the `release` branch will be deleted. It's important to merge back into `develop` because critical updates may have been added to the `release` branch and they need to be accessible to new features. If your organization stresses code review, this would be an ideal place for a pull request.

To finish a `release` branch, use the following methods:

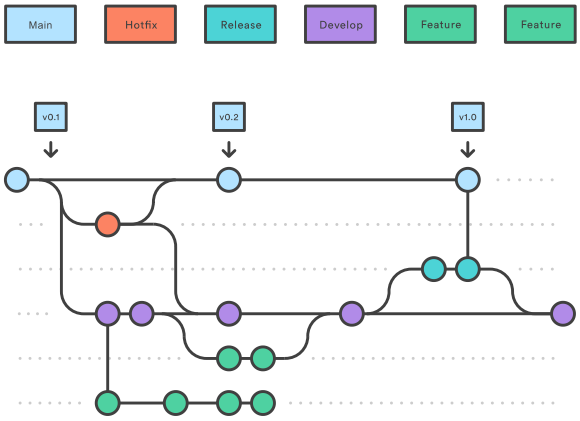
Without the git-flow extensions:

```
git checkout main
git merge release/0.1.0
```

Or with the git-flow extension:

```
git flow release finish '0.1.0'
```

Hotfix branches



Maintenance or `"hotfix"` branches are used to quickly patch production releases. `Hotfix` branches are a lot like `release` branches and `feature` branches except they're based on `main` instead of `develop`. This is the only branch that should fork directly off of `main`. As soon as the fix is complete, it should be merged into both `main` and `develop` (or the current `release` branch), and `main` should be tagged with an updated version number.

Having a dedicated line of development for bug fixes lets your team address issues without interrupting the rest of the workflow or waiting for the next release cycle. You can think of maintenance branches as ad hoc `release` branches that work directly with `main`. A `hotfix` branch can be created using the following methods:

Without the git-flow extensions:

```
git checkout main
git checkout -b hotfix_branch
```

When using the git-flow extensions:

```
$ git flow hotfix start hotfix_branch
```

```
git checkout main
git merge hotfix_branch
git checkout develop
git merge hotfix_branch
git branch -D hotfix_branch
```

```
$ git flow hotfix finish hotfix_branch
```

Example

A complete example demonstrating a Feature Branch Flow is as follows. Assuming we have a repo setup with a `main` branch.

```
git checkout main
git checkout -b develop
git checkout -b feature_branch
# work happens on feature branch
git checkout develop
git merge feature_branch
git checkout main
git merge develop
git branch -d feature_branch
```

In addition to the `feature` and `release` flow, a `hotfix` example is as follows:

```
git checkout main
git checkout -b hotfix_branch
# work is done commits are added to the hotfix_branch
git checkout develop
git merge hotfix_branch
git checkout main
git merge hotfix_branch
```

Summary

Here we discussed the Gitflow Workflow. Gitflow is one of many styles of Git workflows you and your team can utilize.

Some key takeaways to know about Gitflow are:

- The workflow is great for a release-based software workflow.
- Gitflow offers a dedicated channel for hotfixes to production.

The overall flow of Gitflow is:

1. A `develop` branch is created from `main`

2. A `release` branch is created from `develop`

3. `Feature` branches are created from `develop`

4. When a `feature` is complete it is merged into the `develop` branch

6. If an issue in `main` is detected a `hotfix` branch is created from `main`

7. Once the `hotfix` is complete it is merged to both `develop` and `main`

Next, learn about the Forking Workflow or visit our workflow comparison page.

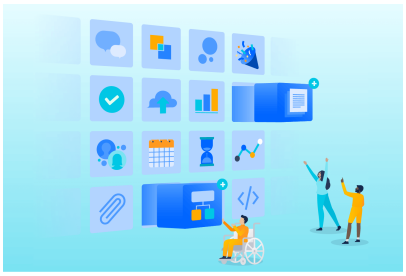
SHARE THIS ARTICLE

NEXT TOPIC

Forking workflow →

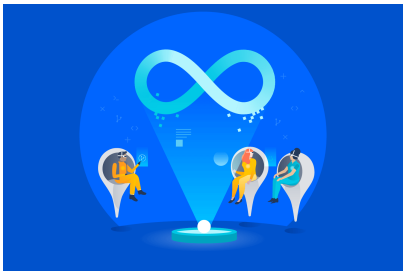
Recommended reading

Bookmark these resources to learn about types of DevOps teams, or for ongoing updates about DevOps at Atlassian.



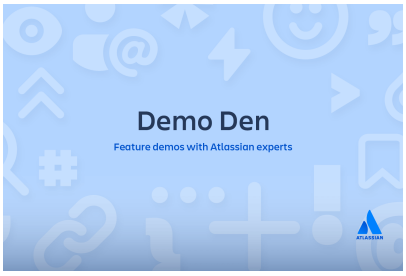
Bitbucket blog

Learn more →



DevOps learning path

Learn more →



How Bitbucket Cloud works with Atlassian Open DevOps

Watch now

Sign up for our DevOps newsletter

Email address

Sign up