**Todos os direitos reservados ao site :** [**https://www.atlassian.com/**](https://www.atlassian.com/)

**Uso apenas para fins educacionais**

Forking Workflow

The Forking Workflow is fundamentally different than other popular Git workflows. Instead of using a single server-side repository to act as the “central” codebase, it gives every developer their own server-side repository. This means that each contributor has not one, but two Git repositories: a private local one and a public server-side one. The Forking Workflow is most often seen in public open source projects.

The main advantage of the Forking Workflow is that contributions can be integrated without the need for everybody to push to a single central repository. Developers push to their own server-side repositories, and only the project maintainer can push to the official repository. This allows the maintainer to accept commits from any developer without giving them write access to the official codebase.

The Forking Workflow typically follows a branching model based on the [Gitflow Workflow](https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow). This means that complete feature branches will be purposed for merge into the original project maintainer's repository. The result is a distributed workflow that provides a flexible way for large, organic teams (including untrusted third-parties) to collaborate securely. This also makes it an ideal workflow for open source projects.

How it works

As in the other [Git workflows](https://www.atlassian.com/git/tutorials/comparing-workflows), the Forking Workflow begins with an official public repository stored on a server. But when a new developer wants to start working on the project, they do not directly clone the official repository.

Instead, they fork the official repository to create a copy of it on the server. This new copy serves as their personal public repository—no other developers are allowed to push to it, but they can pull changes from it (we’ll see why this is important in a moment). After they have created their server-side copy, the developer performs a [git clone](https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-clone) to get a copy of it onto their local machine. This serves as their private development environment, just like in the other workflows.

When they're ready to publish a local commit, they push the commit to their own public repository—not the official one. Then, they file a pull request with the main repository, which lets the project maintainer know that an update is ready to be integrated. The pull request also serves as a convenient discussion thread if there are issues with the contributed code. The following is a step-by-step example of this workflow.

1. A developer 'forks' an 'official' server-side repository. This creates their own server-side copy.
2. The new server-side copy is cloned to their local system.
3. A Git remote path for the 'official' repository is added to the local clone.
4. A new local feature branch is created.
5. The developer makes changes on the new branch.
6. New commits are created for the changes.
7. The branch gets pushed to the developer's own server-side copy.
8. The developer opens a pull request from the new branch to the 'official' repository.
9. The pull request gets approved for merge and is merged into the original server-side repository

To integrate the feature into the official codebase, the maintainer pulls the contributor’s changes into their local repository, checks to make sure it doesn’t break the project, merges it into their local main branch, then pushes the main branch to the official repository on the server. The contribution is now part of the project, and other developers should pull from the official repository to synchronize their local repositories.

It’s important to understand that the notion of an “official” repository in the Forking Workflow is merely a convention. In fact, the only thing that makes the official repository so official is that it’s the public repository of the project maintainer.

Forking vs cloning

It's important to note that "forked" repositories and "forking" are not special operations. Forked repositories are created using the standard [git clone](https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-clone) command. Forked repositories are generally "server-side clones" and usually managed and hosted by a 3rd party Git service like [Bitbucket](https://bitbucket.org/product). There is no unique Git command to create forked repositories. A clone operation is essentially a copy of a repository and its history.

Branching in the Forking Workflow

All of these personal public repositories are really just a convenient way to share branches with other developers. Everybody should still be using branches to isolate individual features, just like in the [Feature Branch Workflow](https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow) and the [Gitflow Workflow.](https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow) The only difference is how those branches get shared. In the Forking Workflow, they are pulled into another developer’s local repository, while in the Feature Branch and Gitflow Workflows they are pushed to the official repository.

Fork a repository

All new developers to a Forking Workflow project need to fork the official repository. As previously stated, forking is just a standard git clone operation. It’s possible to do this by SSH’ing into the server and running git clone to copy it to another location on the server. Popular Git hosting services like Bitbucket, offer repo forking features that automate this step.

Clone your fork

Next each developer needs to clone their own public forked repository. They can do this with the familiar git clone command.

Assuming the use of Bitbucket to host these repositories, developers on a project should have their own Bitbucket account and they should clone their forked copy of the repository with:

git clone https://user@bitbucket.org/user/repo.git

Adding a remote

Whereas other Git workflows use a single origin remote that points to the central repository, the Forking Workflow requires two remotes—one for the official repository, and one for the developer’s personal server-side repository. While you can call these remotes anything you want, a common convention is to use origin as the remote for your forked repository (this will be created automatically when you run git clone) and upstream for the official repository.

git remote add upstream https://bitbucket.org/maintainer/repo

You’ll need to create the upstream remote yourself using the above command. This will let you easily keep your local repository up-to-date as the official project progresses. Note that if your upstream repository has authentication enabled (i.e., it's not open source), you'll need to supply a username, like so:

git remote add upstream https://user@bitbucket.org/maintainer/repo.git

This requires users to supply a valid password before cloning or pulling from the official codebase.

Working in a branch: making & pushing changes

In the developer's local copy of the forked repository they can edit code, commit changes, and create branches just like in other Git workflows:

git checkout -b some-feature # Edit some code git commit -a -m "Add first draft of some feature"

All of their changes will be entirely private until they push it to their public repository. And, if the official project has moved forward, they can access new commits with git pull:

git pull upstream main

Since developers should be working in a dedicated feature branch, this should generally result in a fast-forward merge.

Making a Pull Request

Once a developer is ready to share their new feature, they need to do two things. First, they have to make their contribution accessible to other developers by pushing it to their public repository. Their origin remote should already be set up, so all they should have to do is the following:

git push origin feature-branch

This diverges from the other workflows in that the origin remote points to the developer’s personal server-side repository, not the main codebase.

Second, they need to notify the project maintainer that they want to merge their feature into the official codebase. Bitbucket provides a “pull request” button that leads to a form asking you to specify which branch you want to merge into the official repository. Typically, you’ll want to integrate your feature branch into the upstream remote’s main branch.

Summary

To recap, the Forking Workflow is commonly used in public open-source projects. Forking is a git clone operation executed on a server copy of a projects repo. A Forking Workflow is often used in conjunction with a Git hosting service like Bitbucket. A high-level example of a Forking Workflow is:

1. You want to contribute to an open source library hosted at bitbucket.org/userA/open-project
2. Using Bitbucket you create a fork of the repo to bitbucket.org/YourName/open-project
3. On your local system you execute git clone on https://bitbucket.org/YourName/open-project to get a local copy of the repo
4. You create a new feature branch in your local repo
5. Work is done to complete the new feature and git commit is executed to save the changes
6. You then push the new feature branch to your remote forked repo
7. Using Bitbucket you open up a pull request for the new branch against the original repo at bitbucket.org/userA/open-project

The Forking Workflow helps a maintainer of a project open up the repository to contributions from any developer without having to manually manage authorization settings for each individual contributor. This gives the maintainer more of a "pull" style workflow. Most commonly used in open-source projects, the Forking Workflow can also be applied to private business workflows to give more authoritative control over what is merged into a release. This can be useful in teams that have Deploy Managers or strict release cycles.

Unsure what workflow is right for you? Check out our comprehensive [Git workflow comparison page.](https://www.atlassian.com/git/tutorials/comparing-workflows)

Ready to learn Git?

Try this interactive tutorial.

[Get started now](https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud)