Setting up Gitflow in your repository

When starting a project, you won't have any code files. No problem, just create a Git repository with an empty directory. When finished, you can clone your repository in your system. In this example, we are using a sample GitHub repository, but the procedure applies for any Git repository.

1. Clone the branch in your system using the Windows command prompt:

```
git clone https://github.com/bharatdwarkani/gitflow-example.git
```

2. Switch to the following directory:

```
cd gitflow-example
```

3. Initialize Gitflow in this repository:

```
git flow init
```

You will receive a message stating that no branches exist and prompting you to use common branch names. If you don’t want to change the default branch names, press enter to proceed. When finished, you will have a Gitflow extension initialized in your repository.

> ***Note**: This process has to be done by every developer for any repository they clone in a system. It is not limited to new repositories; it can be used for existing repositories too.*

Master and develop branches

The master and develop branches form the base of any repository in Git. The master branch contains a version of the code that is in production, and the develop branch contains a version that is due to be released in an upcoming version. Execute the following commands in the command prompt to check out the master branch in your system.

```
git checkout master
git push -u origin master
```

```
git push -u origin master
```

You will be prompted for username and password; enter them to proceed. Next, you will need to push the develop branch to a remote repository (i.e. from your system to sync with GitHub). Since the develop branch is only on your local machine, it has to be moved to a remote repository by executing the following commands.

```
git checkout develop
git push origin develop
```

Now you have a repository containing master and develop branches copied from your local machine. This is required only when you first start a project from scratch; otherwise, you could work with one of the following branches.

Feature branch

The feature branch splits from the develop branch and merges back to the develop branch after a feature is complete. The conventional naming of this branch starts with **feature/***. This branch is mostly created and used by developers collaborating with teams. The purpose of the feature branch is to develop small modules of a feature in a project.

You might wonder why developers can't work directly from the develop branch. Why do they need to branch off to a feature branch? To explain this, consider a scenario where you are developing a feature and management decides to drop that feature, as it is no longer required or there is less feasibility of implementing it.

At that time, if you are working in the develop branch directly, it would create a lot of conflicts and possibly break the existing code. Also, to do this you would need to manually delete or comment out code.

Instead, if you branched off a separate feature branch, you could silently discard and delete that branch without affecting the develop branch. Not only does this help develop features, which require the trial-and-error technique, but by using a separate branch you also get an extra level of stability in the develop branch because code from the feature branch undergoes several levels of code reviews and quality assessment before merging into the develop branch.

The lifetime of a feature branch ends once it merges with the develop branch. If multiple developers or teams are working on the same feature, it's easier for them to collaborate by working on a common feature branch.

The following steps show how a feature branch can be created and published using the Gitflow extension from a Windows command prompt.

1. To clone a repository:

```
git clone https://github.com/bharatdwarkani/gitflow-example.git
```

```
cd gitflow-example

git checkout develop

git flow init
```

2. To start a feature branch (the name of the feature branch will be the name of the feature; we are using **feature1** in this example).

```
git flow feature start feature1
```

After execution, the feature1 branch is created, but it exists only on your system and will not be available in the remote GitHub repository. Now you can continue with your development, adding files and modifying the code. When you're done with the feature, you can commit it to your local system and later push it to the remote repository.

3. Once done, the status of the changes can be checked for newly added or modified files.

```
git status
git add .
git commit -am "Your message"
```

4. The following commands publish the feature to the remote repository.

```
git flow publish feature1
git push
```

If you check in the remote repository, a branch with the name **feature/feature1** will be created.

> ***Note**: In the command prompt, the name of the branch you use is* `feature1`*, but Gitflow adds a naming prefix automatically (* `feature/branch` *) as a convention. When specifying a branch name in Git commands, you need to use the full branch name (* `feature/feature1` *), but in Gitflow commands the general prefix (* `feature/` *) need not be specified.*

5. Once a feature is complete and the code has been reviewed, you can complete your work in a branch by issuing the command below. Upon execution, the code will be merged to the development branch automatically and the feature branch will be deleted from the remote repository.

```
git flow finish feature1
```

6. If you need to delete a branch, you can execute: `git branch -d feature/feature1`

> ***Note**: If multiple developers are coordinating on a feature, they need to follow the previous steps for cloning, with one caveat: One of the developers has to create and publish a feature branch, which might be empty, so that the others can work collaboratively. If a new developer needs to work, he or she can follow the same process by modifying the following command.*

`git flow feature track feature1` instead of `git flow feature start feature1`

Apart from this, there is one more branch called `bugfix`. It has a workflow similar to the `feature` branch, but it is used to fix a bug.

Release branch

The release branch derives from the develop branch and merges back into the develop and master branches after completion of a release. By convention, the naming of this branch starts with `release/*` . This branch is created and used when features are completed and finalized for a versioned release.

Why can't we directly release from the develop branch? Because the sole purpose of the release branch is to isolate a version of a release that is final but needs some quality assistance and stability from the upcoming version. If we branch off from the release branch, then other developers who are assigned to develop features for an upcoming release and are not involved in the release stability process can continue to development and merge their features into the develop branch without waiting on or affecting the current release process. The release branch helps isolate the development of an upcoming version and the current release.

The release branch's lifetime ends when a particular version of a project is released. Once this branch merges into the develop and master branches, it can be deleted. And

once you have done this, you can tag a master branch with a particular release version —let's say **v1.0**—to create a historical milestone.

The following example explains how a release branch can be created and published using the Gitflow extension from the command prompt.

1. Start a release branch.

```
git checkout develop
git pull
git flow release start release1
```

2. Commit newly added or modified changes and push to the remote repository.

```
git add .
git commit -m "Your message"
```

```
git commit -am "Your message"
git flow publish release1
git push
```

3. Merge changes to the develop branch.

```
git checkout develop
git merge release/release1
```

4. After a release, merge changes to the master branch.

```
git checkout release/release1
git pull
git flow release finish release1
```

-or-

```
git flow release finish -m "Your message" "release1"
git checkout master
git push --all origin
```

Hotfix branch

The hotfix branch is derived from the master branch and merged back after completion to the develop and master branches. By convention, the name of this branch starts with `hotfix/*` . This branch is created and used after a particular version of product is released to provide critical bug fixes for the production version.

The reason we do this is because one problem you might face when branching off from the develop branch is that some of your developers would have already started work for the upcoming release while you are in the middle of the current release. Your release would contain the next version of features, which are not finalized, but you only need to provide bug fixes for the current version. Instead of branching off from develop branch, you can branch off from the master branch, as that branch contains only the current version of the code in production. This way, branching off from the master branch will not affect your production or development version of the product.

The hotfix branch can be deleted once a critical fix for the current production version is released and merged with the master and development branches. Once you have done this, you can again tag the master branch with an iterative subversion of the release; let's say v**1.1**.

This example shows how the `hotfix1` branch can be created and published using the Gitflow extension from a command prompt.

1. Start a new hotfix branch and commit changes after modifications.

```
git checkout develop
git flow hotfix start hotfix1
git status
git add .
git commit -am "Your message"
```

2. Publish the branch to the remote repository.

3.

```
git flow publish hotfix1
```

Merge changes to remote repository.

```
git flow hotfix finish hotfix1
```

-or-

```
git flow hotfix finish -m "Your message" "hotfix1"
git status


git checkout master
git push --all origin
```

> ***Note**: All commands starting with "**git flow**" are based on the Gitflow extension. Actual Git commands don't have flow keywords in them. They start only with "**git**."*