## **What Is GitFlow?**

[GitFlow](http://nvie.com/posts/a-successful-git-branching-model/) is a branching model for Git, created by Vincent Driessen. It has attracted a lot of attention because it is very well suited to collaboration and scaling the development team.

## **Key Benefits**

### **Parallel Development**

One of the great things about GitFlow is that it makes parallel development very easy, by isolating new development from finished work. New development (such as features and non-emergency bug fixes) is done in **feature branches**, and is only merged back into main body of code when the developer(s) is happy that the code is ready for release.

Although interruptions are a BadThing(tm), if you are asked to switch from one task to another, all you need to do is commit your changes and then create a new feature branch for your new task. When that task is done, just checkout your original feature branch and you can continue where you left off.

### **Collaboration**

Feature branches also make it easier for two or more developers to collaborate on the same feature, because each feature branch is a sandbox where the only changes are the changes necessary to get the new feature working. That makes it very easy to see and follow what each collaborator is doing.

### **Release Staging Area**

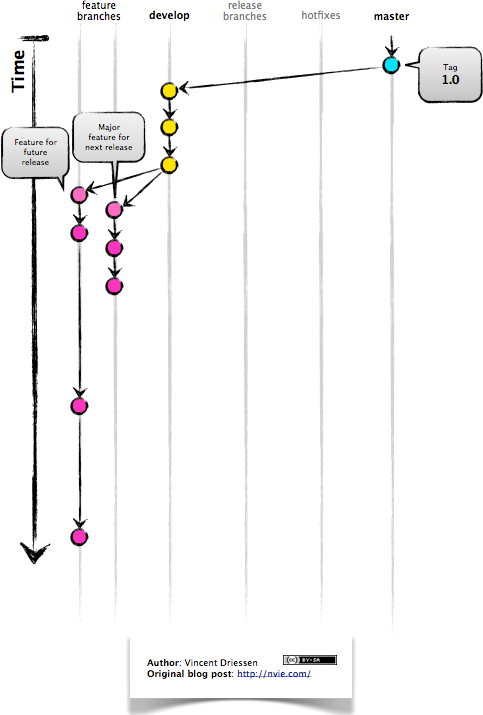
As new development is completed, it gets merged back into the **develop branch**, which is a staging area for all completed features that haven’t yet been released. So when the next release is branched off of **develop**, it will automatically contain all of the new stuff that has been finished.

### **Support For Emergency Fixes**

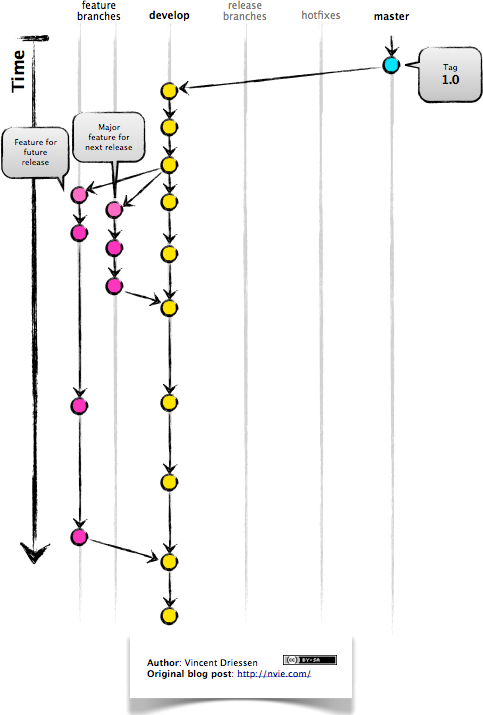
GitFlow supports **hotfix branches** - branches made from a tagged release. You can use these to make an emergency change, safe in the knowledge that the hotfix will only contain your emergency fix. There’s no risk that you’ll accidentally merge in new development at the same time.

## **How It Works**

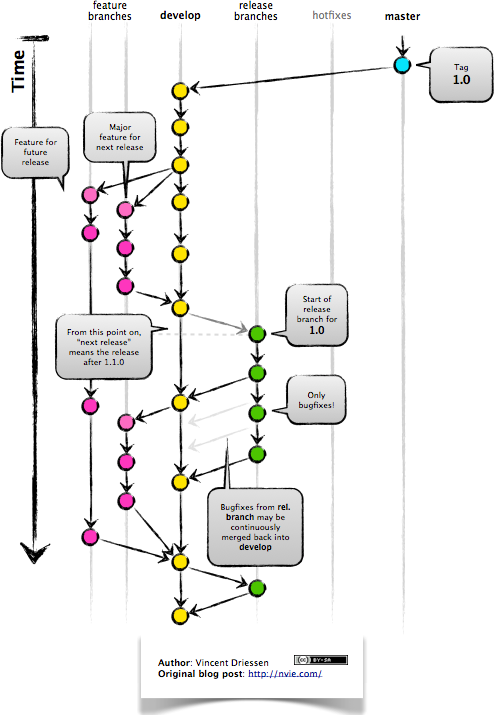
New development (new features, non-emergency bug fixes) are built in **feature branches**:



Feature branches are branched off of the **develop branch**, and finished features and fixes are merged back into the **develop branch** when they’re ready for release:

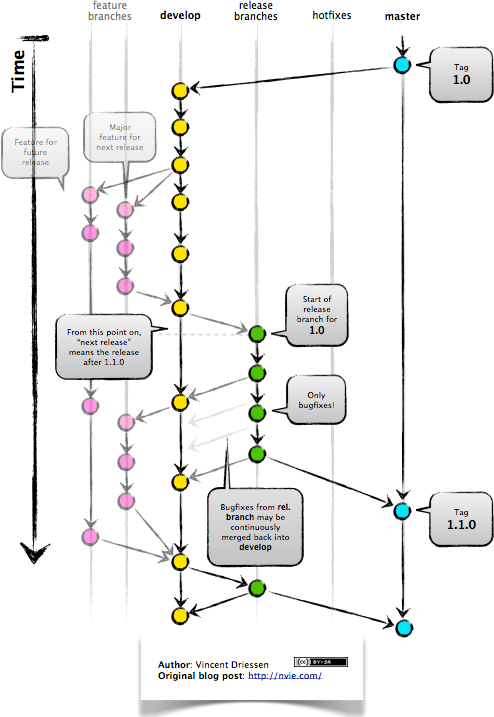


When it is time to make a release, a **release branch** is created off of **develop**:



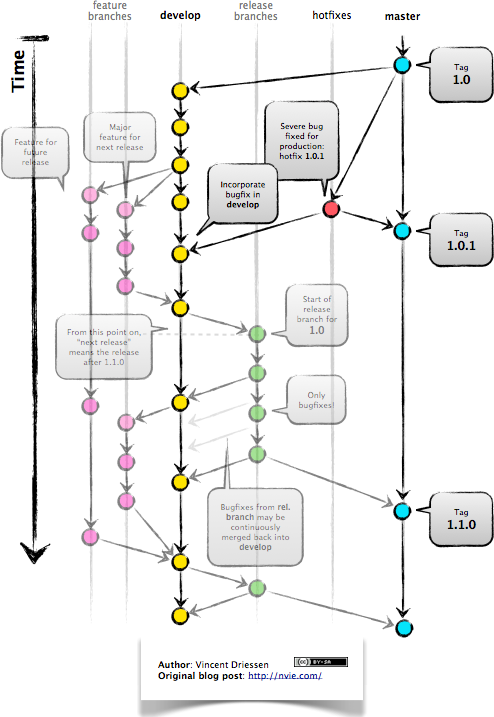
The code in the **release branch** is deployed onto a suitable test environment, tested, and any problems are fixed directly in the release branch. This **deploy -> test -> fix -> redeploy -> retest** cycle continues until you’re happy that the release is good enough to release to customers.

When the release is finished, the **release branch** is merged into **master** **and** into **develop** too, to make sure that any changes made in the **release branch** aren’t accidentally lost by new development.



The **master branch** tracks released code only. The only commits to **master** are merges from **release branches** and **hotfix branches**.

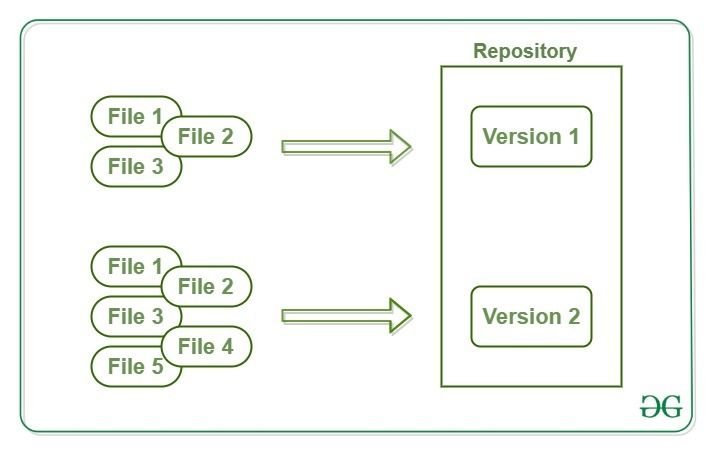
**Hotfix branches** are used to create emergency fixes:



They are branched directly from a tagged release in the **master branch**, and when finished are merged back into both **master** and **develop** to make sure that the hotfix isn’t accidentally lost when the next regular release occurs.

# Repository

Repositories in **GIT** contain a collection of files of various different versions of a Project.



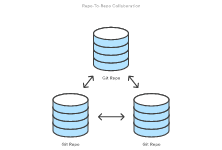
# Clone

Cloning a local or remote repository

Cloning a bare repository

Using shallow options to partially clone repositories

Git URL syntax and supported protocols



# Fork

A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

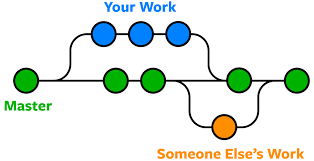
# Branch

**In Git, branches are a part of your everyday development process. Git branches are effectively a pointer to a snapshot of your changes. When you want to add a new feature or fix a bug—no matter how big or how small—you spawn a new branch to encapsulate your changes.**

**Master Definition:**

In **Git**, "**master**" is a naming convention for a branch. The link for our repository for this group assignment is found <https://github.com/EmadAbdelhamidNJIT/IS601Ass1.git>

After cloning the Master (downloading) a project from a remote server, the resulting local repository has a single local branch: the so-called "**master**" branch. This means that "**master**" can be seen as a repository's "default" branch.



**Remove/Add/Remove/Show:**

The "remote" repository that is the source of a fetch or pull operation. This parameter can be either a URL or the name of a remote (see the section [REMOTES](https://git-scm.com/docs/git-fetch#REMOTES)).

“Git Add”: This command updates the index using the current content found in the working tree, to prepare the content staged for the next commit. It typically adds the current content of existing paths as a whole, but with some options it can also be used to add content with only part of the changes made to the working tree files applied or remove paths that do not exist in the working tree anymore.

“Git Remove”: A user can remove a branch by simply typing in ‘git push origin :<location>’

“Git Show”: Is the default for ‘git log’…git-show is a command line utility that is used to view expanded details on Git objects such as blobs, trees, tags, and commits. git-show has specific behavior per object type. Tags show the tag message and other objects included in the tag. Trees show the names and content of objects in a tree.

**Git Status:**

Define Status: **git status** command displays the state of the working directory and the staging area. It lets you see which changes have been staged, which haven't, and which files aren't being tracked by **Git**.

Only working on the project, after every ‘git add –all’ commit I checkout the status of my branch, if I committed the message using the wrong name it returned a message ‘untracked files ~$ster Branch.docx’ I realize the misspelling deviated the commit.