What Is GitFlow?

What Is GitFlow?

GitFlow 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.

GitFlow 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.

- **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.

GitFlow Key Benefits

- **Release Staging Area**

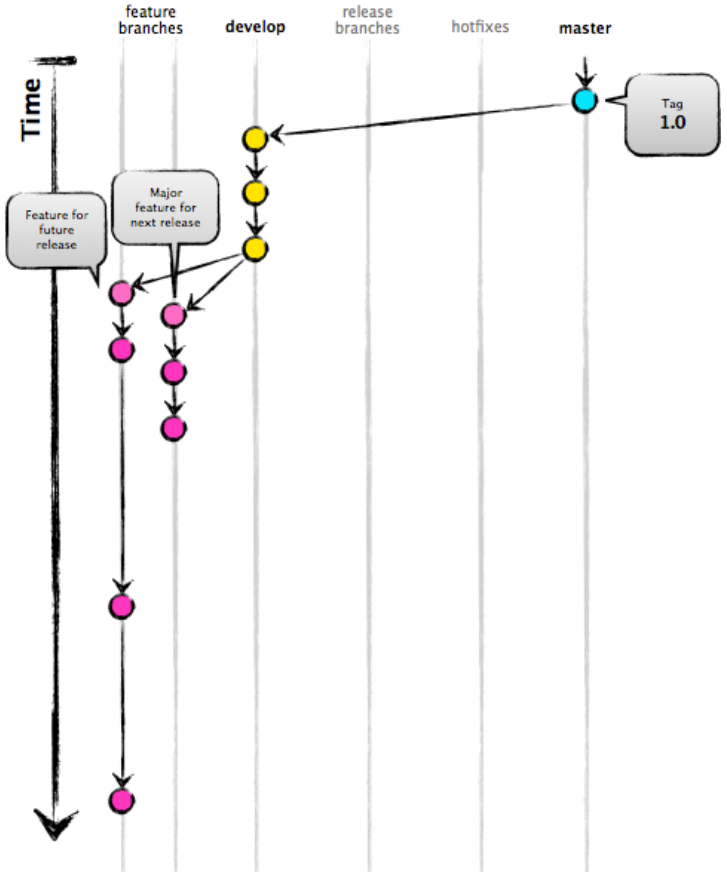
As new development is completed, it gets merged back into the develop branch, which is a staging area for all completed features that haven't yet been released. So when the next release is branched off of develop, it will automatically contain all of the new stuff that has been finished.

- **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.

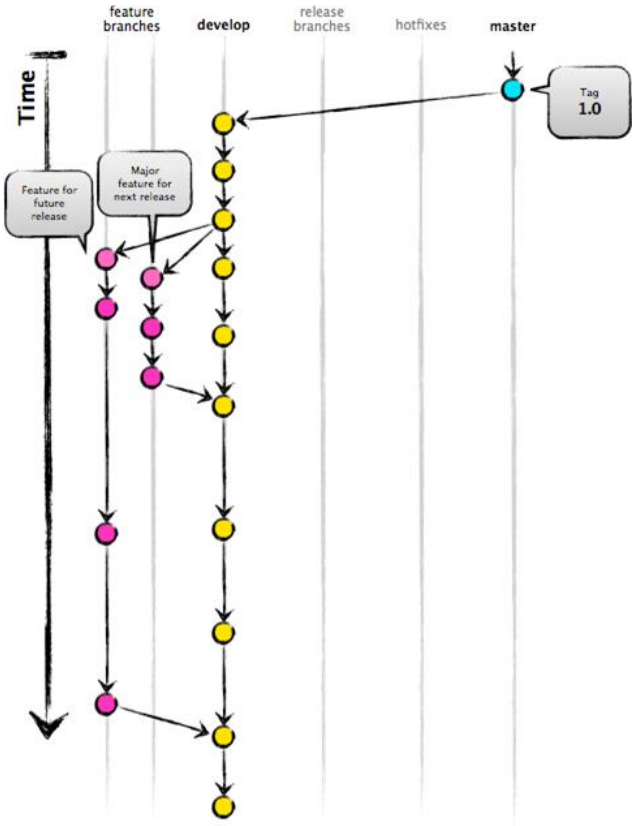
How GitFlow works

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



How GitFlow works

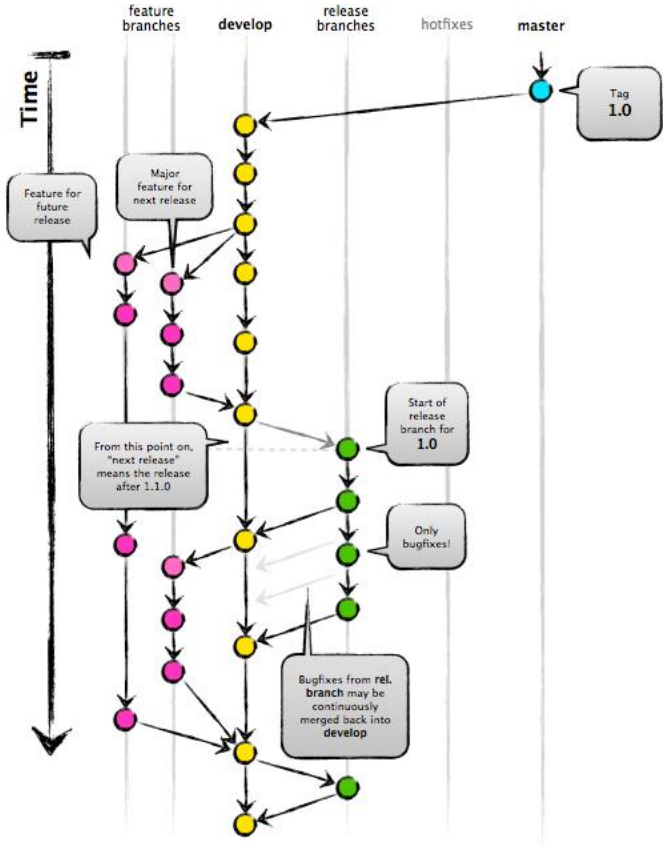
Feature branches are branched off of the *develop branch*, and finished features and fixes are merged back into the *develop branch* when they're ready for release:



How GitFlow works

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

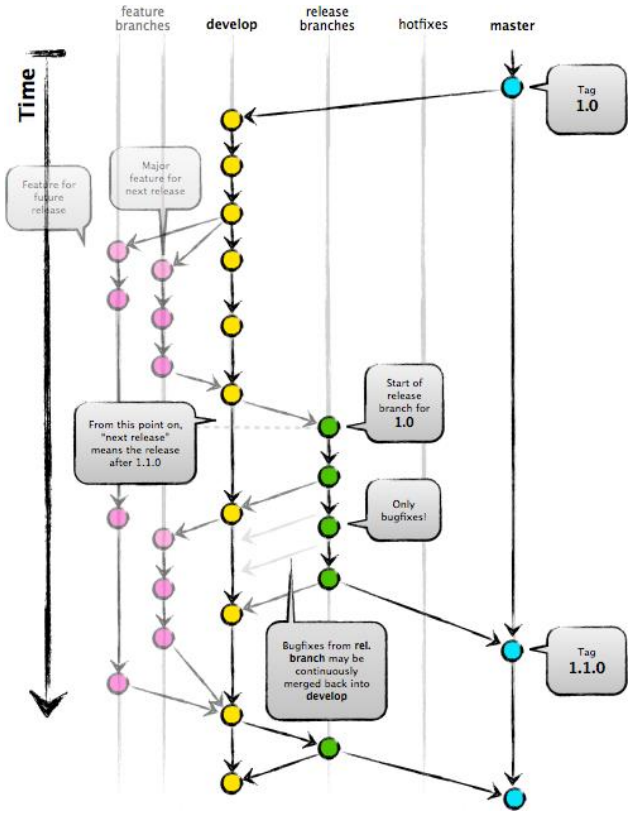
The code in the *release branch* is deployed onto a suitable test environment, tested, and any problems are fixed directly in the release branch. This *deploy -> test -> fix -> redeploy -> retest* cycle continues until you're happy that the release is good enough to release to customers.



How GitFlow works

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.

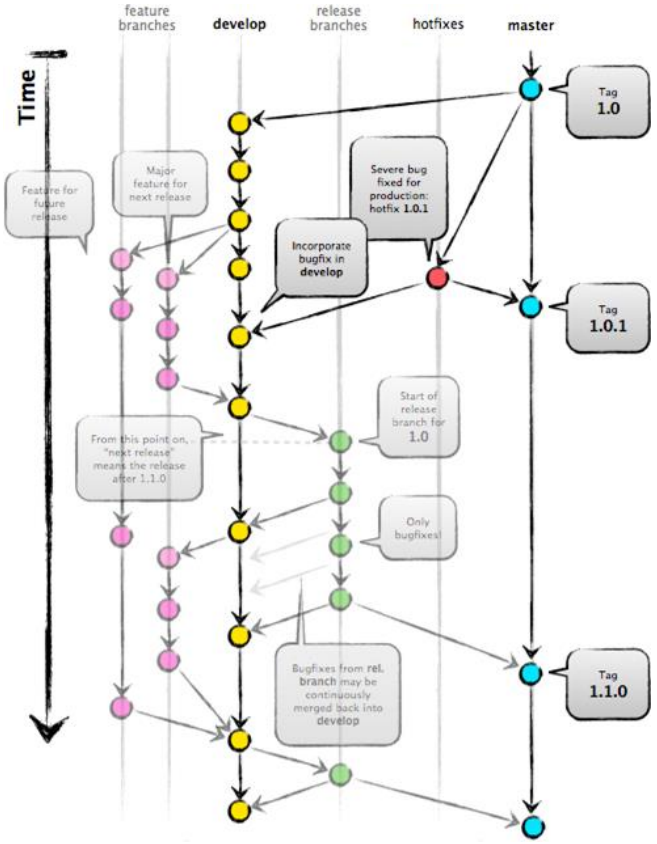
The *master branch* tracks released code only. The only commits to *master* are merges from *release branches* and *hotfix branches*.



How GitFlow works

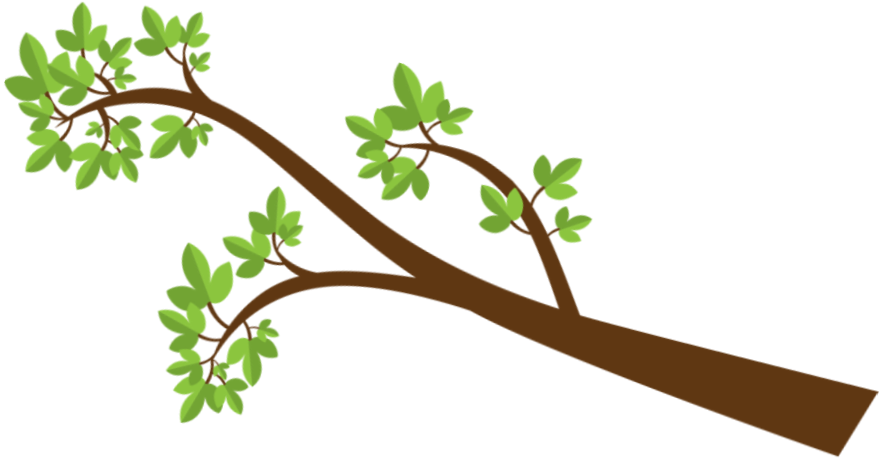
Hotfix branches are used to create emergency fixes:

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.

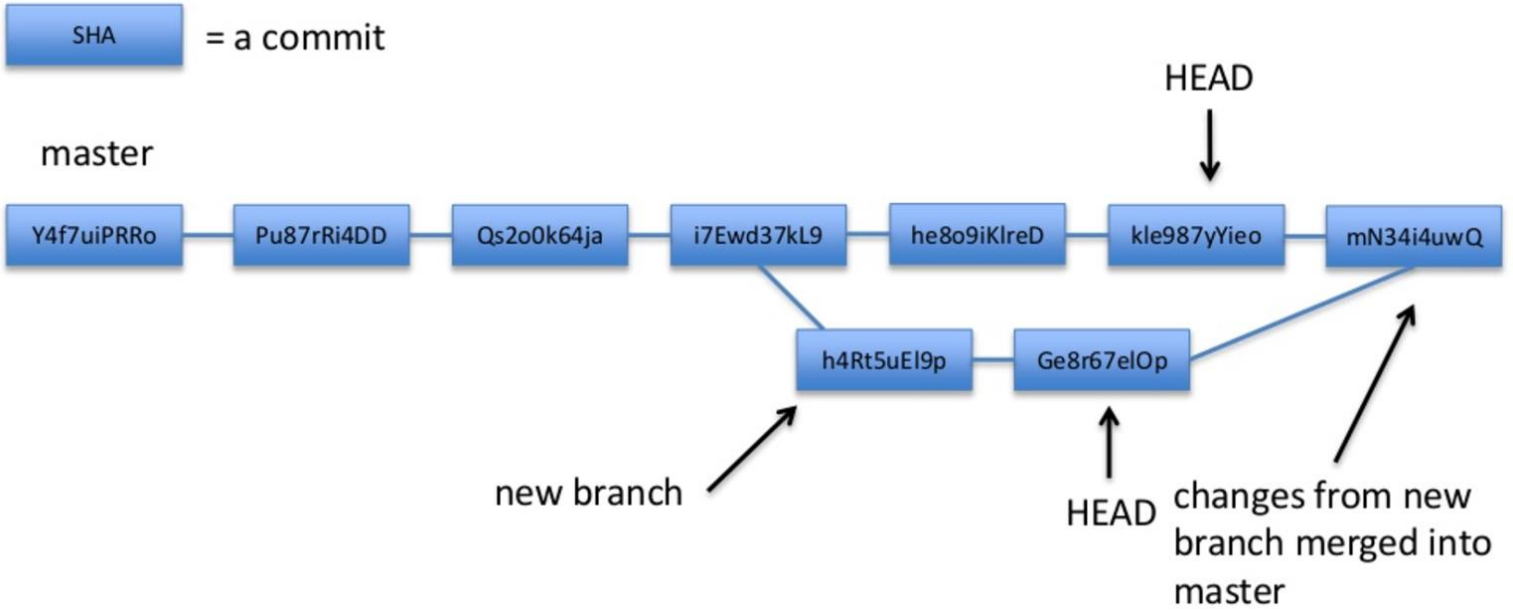


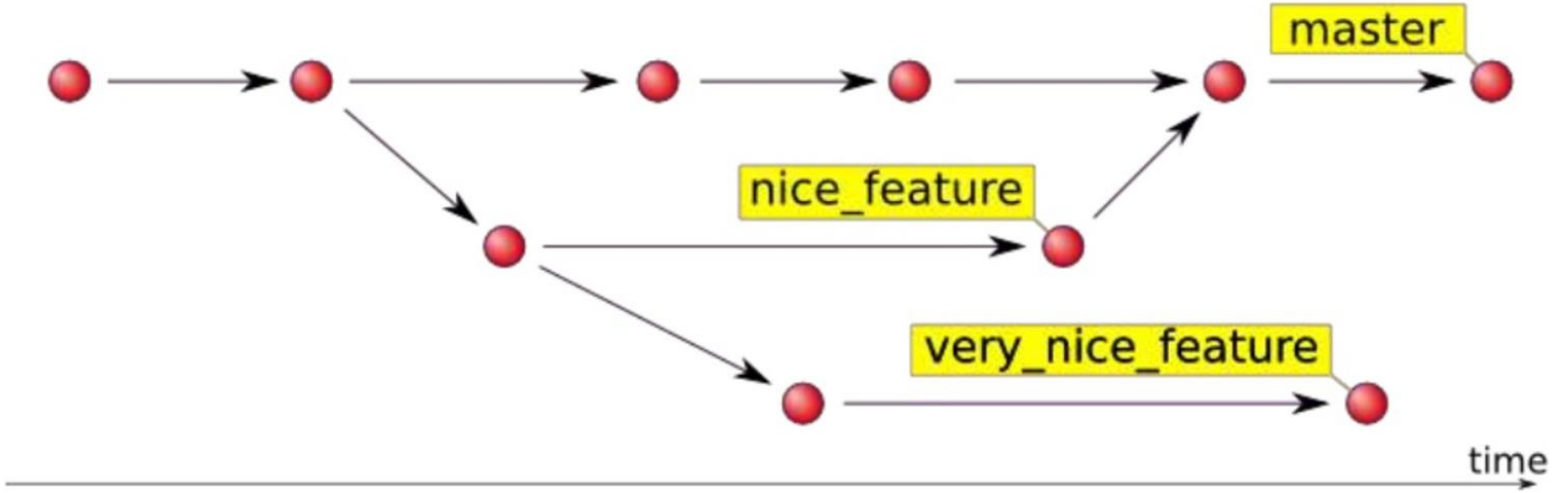
Branching

- allows one to try new ideas
- If an idea doesn’t work, throw away the branch. Don’t have to undo many changes to master branch
- If it *does* work, merge ideas into master branch.
- There is only one working directory



Branching and merging example




master
nice_feature
very_nice_feature
time

In which branch am I?

git branch

```
[dolanmi]$ git branch
* master
```

How do I create a new branch?

git branch new_branch_name

```
[dolanmi]$ git branch
* master
  new_feature
```

Note: At this point, both HEADs of the branches are pointing to the same commit (that of master)

How do I switch to new branch?

git checkout new_branch_name

```
[dolanmi]$ git checkout new_feature
Switched to branch 'new_feature'
[dolanmi]$ git branch
  master
* new_feature
```

At this point, one can switch between branches, making commits, etc. in either branch, while the two stay separate from one another.

Note: In order to switch to another branch, your current working directory must be clean (no conflicts, resulting in data loss).

Comparing branches

git diff first_branch..second_branch

```
[dolanmi]$ git diff master..new_feature
diff --git a/file1.txt b/file1.txt
index 5626abf..1684a0f 100644
--- a/file1.txt
+++ b/file1.txt
@@ -1 +1 @@
-one
+new information
```

How do I merge a branch?

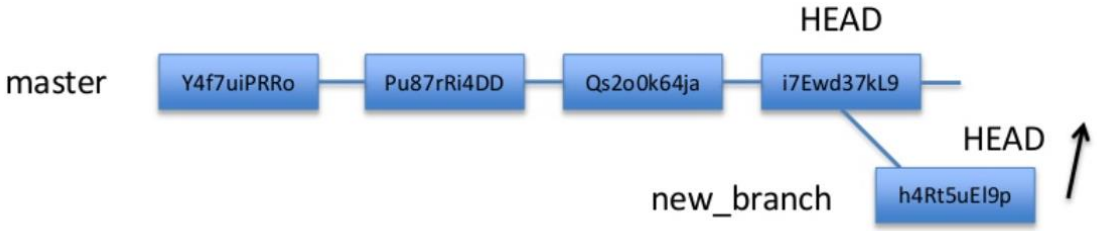
From the branch into which you want to merge another branch….

git merge branch_to_merge

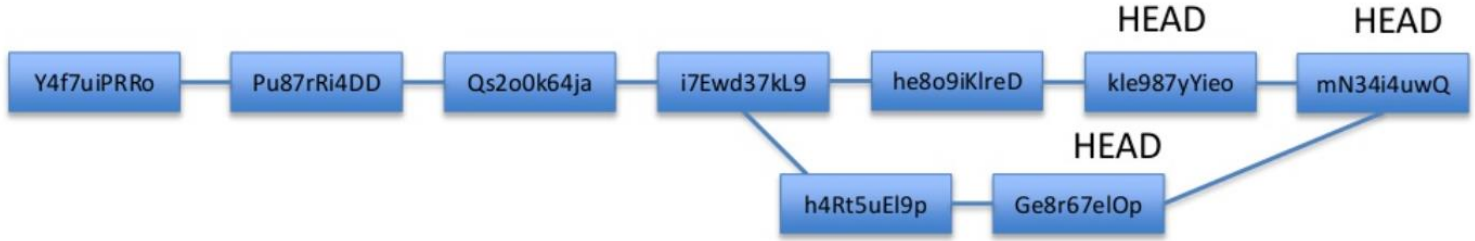
```
[dolanmi]$ git branch
* master
  new_feature
[dolanmi]$ git merge new_feature
Updating 3789cd3..1214807
Fast-forward
 file1.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
[dolanmi]$ git diff master..new_feature
[dolanmi]$
```

Note: Always have a clean working directory when merging

"fast-forward" merge occurs when HEAD of master branch is seen when looking back

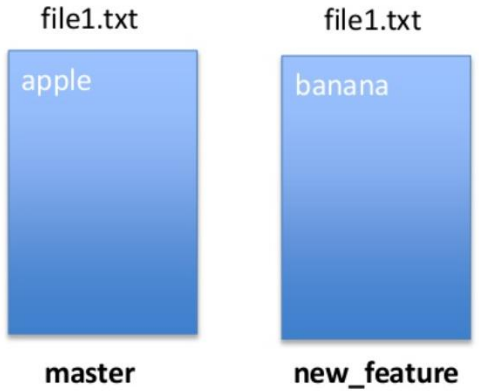


"recursive" merge occurs by looking back and combining ancestors to resolve merge



Merge conflicts

What if there are two changes to same line in two different commits?




```
[dolanmi]$ git merge new_feature
Auto-merging file1.txt
CONFLICT (content): Merge conflict in file1.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Resolving merge conflicts

Git will notate the conflict in the files!

```
<<<<<<< HEAD
apple
=======
banana
>>>>>>> new_feature
```

Solutions:

1. Abort the merge using **git merge –abort**
2. Manually fix the conflict
3. Use a merge tool (there are many out there)

Graphing merge history

git log --graph --oneline --all --decorate

```
[dolanmi]$ git log --graph --oneline --all --decorate
*   7367e1e (HEAD -> master) fix merge conflict
|\
| * b4f09a5 (new_feature) add banana
* | df043c1 add apple
|/
* 1214807 new information added
* 3789cd3 file3.txt
* 6bfebcd new dir
* 730c6bd files
* 48f1ecf c
* 60f1c1a another message yet
* d685ff9 another message
* 6e073c6 message
```

Tips to reduce merge pain

- merge often
- keep commits small/focused
- bring changes occurring to master into your branch frequently (“tracking”)

Q&A