**What is Git Flow?**

1. A set of guidelines developers can follow when using version control
2. Referred to as a “Branching Model”
3. Not rules, but guidelines which are not set in stone(to be very difficult or impossible to change)

**How it works?**

1. Git flow work with the central repository (Master).
2. Developers in your team will clone the central repository and work locally and then push there branches to the central repository.
3. Two branches used to record project history, “Master” and “Develop”
4. Develop serves as an integration branch for features for devs to work on
5. Master branch stores the official release history



**Release branches?**

1. Release branches are created by forking “Develop”
2. Senior Dan will create a “Release branch”
3. The “Release branch” will contain a pre-determined amount of features
4. The “Release branch” should be deployed to Staging sever for QA testing **(Quality Assurance)**

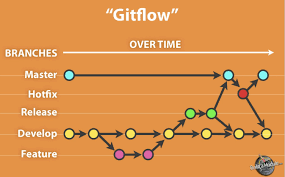
***Quality Assurance***

* *QA includes activities that ensure the implementation of processes, procedures and standards in context to verification of developed software and intended requirements.*
* *Focuses on processes and procedures rather than conducting actual testing on the system.*
* *Process-oriented activities.*
* *Preventive activities.*

1. Any bugs, needs to be addressed on the “release branch”
2. The “release branch” will have to be merged back into “develop” as well as “master”
3. You should then tag “master” with a version number

**Managing Hotfixes**

1. Hotfixes are defined as minor fixes to the project
2. Fork “master” to create a new “hotfix” branch
3. Commit code to the “hotfix” branch
4. The “hotfix” branch, once tested, must be merged into “master” and “develop”
5. The “master” branch should be tagged again and deployed



Now just to summarize these three different types of branches, a release branch is a holding area for code that can be tested and fixed before it gets pushed to production. It gets branched directly from the develop branch; and when it's done, it gets pushed to production or the master branch, it gets merged back into the development branch, and then it gets deleted. The purpose of a hotfix branch is to take a bug that we find on the master branch that needs immediate attention and do a branch directly from master, make the bug fix, and then push that bug fix back to master as a new release and push it back to development to get incorporated with the next major release, and then it's deleted.

**Summary of the concept**

The general idea of git-flow is to use the following branch structure in your repository:

* **Development branch** (usually called ‘develop’)  
  This is your main development branch where all the changes destined for the next release are placed, either by directly committing small changes or by merging other branches (e.g. feature branches) into this branch.
* **Production branch** (usually called ‘master’)  
  This branch represents the latest released / deployed codebase. Only updated by merging other branches into it.
* **Feature branches** (usually prefixed with ‘feature/’)  
  When you start work on anything non-trivial (important), you create a feature branch. When finished, you’ll merge this branch back into the development branch to queue it for the next release.
* **Release branches** (usually prefixed with ‘release/’)  
  When you’re about to package a new release, you create a release branch from the development branch. You can commit to it during your preparation for a release, and when it’s ready to be deployed, you merge it into both the development branch and the master branch (to indicate that the release has been deployed).
* **Hotfix branches** (usually prefixed with ‘hotfix/’)  
  If you need to patch the latest release without picking up new features from the development branch, you can create a hotfix branch from the latest deployed code in master. Once you’ve made your changes, the hotfix branch is then merged back into both the master branch (to update the released version) and the development branch (to make sure the fixes go into the next release too)

SourceTree helps you utilise these branches via git-flow actions which we will describe below.

**Version control** is a system that records changes to a file or set of files so that you can recall specific versions later. For the examples in this book, you will use software source code as the files being version controlled, though in reality you can do this with nearly any type of file on a computer.

If you are a graphic or web designer and want to keep every version of an image or layout (which you would most certainly want to), a Version Control System (VCS) is a very wise thing to use. It allows you to revert selected files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more. Using a VCS also generally means that if you screw things up or lose files, you can easily recover. In addition, you get all this for very little overhead.