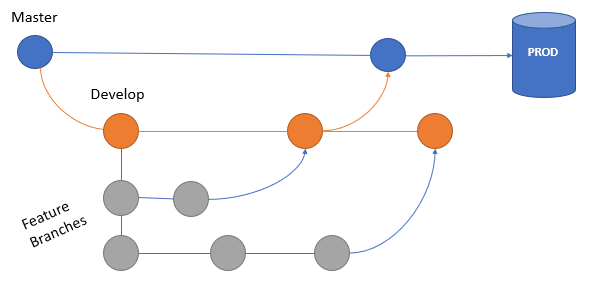
**What is Feature Branch Workflow**

The core idea behind the Feature Branch Workflow is that all feature development should take place in a dedicated branch instead of the master branch. This encapsulation makes it easy for multiple developers to work on a particular feature without disturbing the main codebase. It also means the master branch will never contain broken code, which is a huge advantage for continuous integration environments.

The Feature Branch Workflow assumes a central repository, and master represents the official project history. Instead of committing directly on their local master branch, developers create a new branch every time they start work on a new feature

In addition, feature branches can (and should) be pushed to the central repository. This makes it possible to share a feature with other developers without touching any official code. Since master is the only “special” branch, storing several feature branches on the central repository doesn’t pose any problems. Of course, this is also a convenient way to back up everybody’s local commits. The following is a walk-through of the life-cycle of a feature branch.

**How looks a Feature Branching Strategy?**



**How Semantic Versioning can help us?**

SemVer (Semantic Versioning) provides a standardized format for conveying versioning information as well as guidelines for usage. Software release information becomes meaningful, a glance at the number describes what to expect when consuming a new update or new piece of software.

SemVer is a 3-component system in the format of X.Y.Z where:

* X stands for a Major version
* Y stands for a Minor version
* Z stands for a patch

Some rules of SemVer are:

1. Each increment happens numerically with an increment of 1.
2. When you fix a bug and your code stays backwards compatible, you increase the patch component:

* v0.0.3 // Before bugfix
* v0.0.4 // After bugfix

1. When you add functionality and your code stays backwards compatible, you increase the minor component and reset the patch component to zero:

* v0.2.4 // Before addition of new functionality
* v0.3.0 // After addition of new functionality

1. When you implement changes and your code becomes backwards incompatible, you increase the major component and reset the minor and patch components to zero:

* v7.3.5 // Before implementing backwards incompatible changes
* v8.0.0 // After implementing backwards incompatible changes

Some unofficial conventions are:

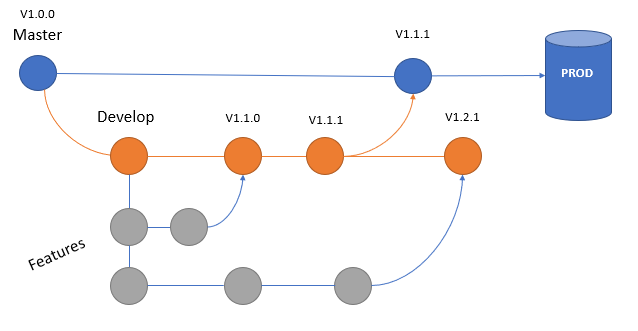
* 0.Y.Z (a major component of zero) is generally used for initial development
* When your code is used in production, you should increase to version 1.0.0 (a major component of one or higher)

Platforms like Git or Bitbucket, can help us with our projects versioning by using tags to define the version of our project in every branch.

Example of Semantic Versioning:

* v0.0.0 // New project
* v0.1.0 // Add some functionality
* v0.2.0 // Add other new functionality
* v0.2.1 // Fix bug
* v0.3.0 // Add some functionality
* v0.3.1 // Fix bug
* v0.3.2 // Fix bug
* v0.4.0 // Add some functionality
* v0.4.1 // Fix bug
* v1.0.0 // Code is being used in production
* v1.1.0 // Add some functionality
* v1.2.0 // Add other new functionality
* v1.2.1 // Fix bug
* v2.0.0 // Implement changes that causes public API of code to become backwards incompatible

**How looks a Versioned Branching Strategy?**



Some benefits of versioning our projects are

* Keep a semantic historical track of a component
* Know which version of a component is no longer backwards compatible
* Avoid dependency hell when using a component in different places
* Allow a component to be distributed correctly with package managers

**Hotfix Branches**

Sometimes when a critical bug in a production version must be resolved immediately, we need to create a hotfix branch from the corresponding tag on the master branch that marks the production version. This is necessary so that the team members on the develop branch can continue with the activities, while another person is preparing a quick production fix in the Hotfix branch. Here an example:

