Workflow version control

This document details the workflow within Thywin, search engine project. It covers usage of GIT, Versioning and branching.

The following tools are used:

* GITHub for Windows

## New Features

Each new feature will first be discussed internally with the development team. The development team will prioritize these features in collaboration with the customer. Features with a high priority will be developed before features with lower priority. A developer will be assigned to a feature based on availability and preference.

The workflow is accordingly:

1. Feature Idea gets formulated.
2. Idea discussed with customer and prioritized.
3. An initial implementation will be made outside the main development branch (*Details on this can be found in the Version Control subject*).
4. Initial implementation will be presented to customer.
5. Proper implementation will be made for integration within the main development branch for merge into master/release branches.

## Bug Fixes

Once a bug has been reported, a how to reproduce will be written up. Once its cause has been identified, it will be fixed and merged into the active development branch. In case of a product breaking or critical bug, a hotfix will be applied containing just the fix to the release & master branches.

## Version Control

#### Versioning

Thywin makes use of Semantic Versioning 2.0.0. (<http://semver.org/>).  This means our versioning format is MAJOR.MINOR.PATCH.

We will increment the:

1. MAJOR version when you make incompatible API changes,
2. MINOR version when you add functionality in a backwards-compatible manner, and
3. PATCH version when you make backwards-compatible bug fixes.

## GIT Usage

GIT will be our version control system. All code and documents will be stored in the GIT repository.

For the usage of GIT we have a simple set of rules:

1. All code pushed to the repository should compile.
2. All commits contain a proper description on what has been changed.
3. Only files related to the project can be pushed to the repository.
4. All code/files/folders are structured in agreement with the Thywin Coding Standards (see HAP-plan).

#### GIT Branching Model

We use a branching model similar to the model published on <http://nvie.com/posts/a-successful-git-branching-model/>. This means that there won’t be any commits to the master directly. Instead, there is a development branch. This branch will be the base for all features being developed.

Once a new version has been marked, the development branch will make a pull request to the master branch, upon which the development branch will receive a feature freeze.

When a new release has been merged into the master, it will receive a tag with a correct version number.

Any features being developed are based off the development branch. Once a feature has been completed, it will be merged back into the development branch.

#### Branches

|  |  |
| --- | --- |
| **Name** | **Type** |
| Master | Develop and Hotfix branches make pull requests to this branch - no other actions allowed. Should always be tagged with a version number. |
| HotFix-\* | Only bases off Master. No merge into |
| Develop | Active development branch. |
| Feature-\* | Based off the latest development branch. Can be rebased from latest development branch only. |

## Changelogs

Every commit should have a descriptive title and description in compliance with the Version control standards. By doing this, we are able to use our commits as a changelog.

Every release, public and private/closed, should have a changelog attached detailing what has changed and which are the new features. The changelog should have each entry clearly marked with a “fixed:”, “changed:”, “removed:” or “new:” tag in front for easy identification by users and developers.