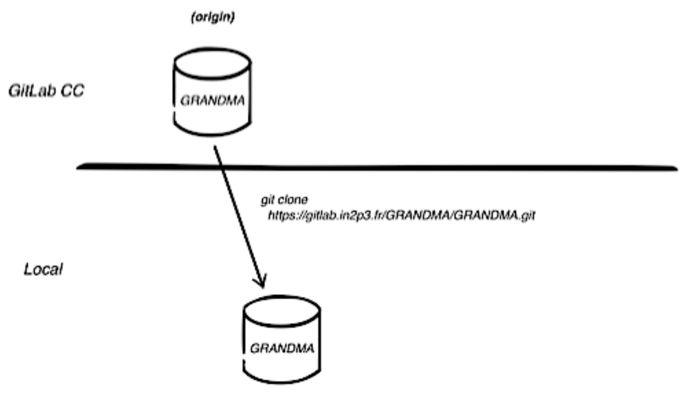
GRANDMA

GRANDMA development workflow

This document describes the development workflow followed by the GRANDMA collaboration.

Environment setup



Cloning GRANDMA GitLab repository

Clone GRANDMA repository to your local working area. From your command line:

```
$> git clone git@gitlab.in2p3.fr:GRANDMA/GRANDMA.git GRANDMA.git
$> cd GRANDMA.git
```

Installing the gitflow script

> GRANDMA's workflow is based on a light Driessen's branching model; you'll need to install once a git extension to provide high-level git support. We are using `git-flow (AVH Edition)`, a fork of original Driessen script: petervanderdoes/gitflow-avh

See the gitflow-avh Wiki for up-to-date Installation Instructions. In a nutshell:

- on Mac OS X:

  ```
  $> brew install git-flow-avh
  ```

- on other *nix:

  ```
  $> wget --no-check-certificate -q \
     https://raw.githubusercontent.com/petervanderdoes/gitflow-avh/develop/contrib/gitflow-installer.sh \
     && sudo bash gitflow-installer.sh install stable
  $> rm gitflow-installer.sh
  ```

- on Windows:

  ```
  See [Installing gitflow-avh on Windows](https://github.com/petervanderdoes/gitflow-avh/wiki/Installing-on-Windows)
  ```

Check:

```
$> git flow version
1.11.0 (AVH Edition)

$> git flow help
usage: git flow <subcommand>

Available subcommands are:
   init      Initialize a new git repo with support for the branching model.
   feature   Manage your feature branches.
   bugfix    Manage your bugfix branches.
   release   Manage your release branches.
   hotfix    Manage your hotfix branches.
   support   Manage your support branches.
   version   Shows version information.
```

```
   config    Manage your git-flow configuration.
   log       Show log deviating from base branch.

Try 'git flow <subcommand> help' for details.
```

Initializing gitflow

You'll only need to do this once, just after cloning:

```
$> cd GRANDMA.git
$> git flow init -d
Using default branch names.

Which branch should be used for bringing forth production releases?
   - develop
   - master
Branch name for production releases: [master]

Which branch should be used for integration of the "next release"?
   - develop
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [/Some/Where/GRANDMA/GRANDMA.git/.git/hooks]
```

`-d` option selects defaults values.

Verify that the two main branches are there:

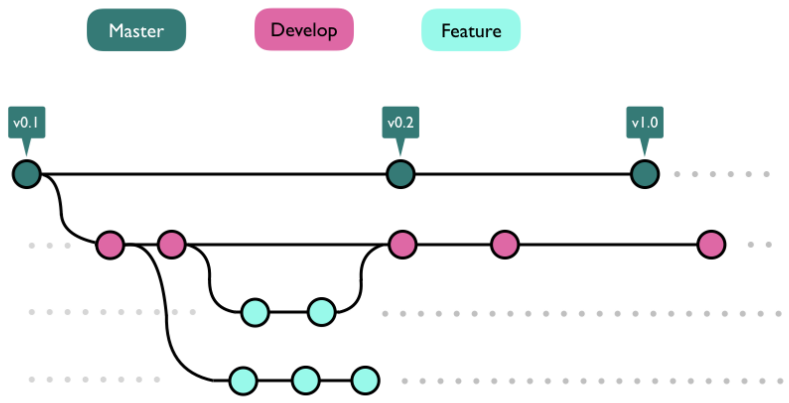
```
$> git branch
* develop
  master
```

The *develop* branch is the selected branch (with its *") where most of the work will happen, and the *master* branch will keep track of production-ready code. More details inDevelopment workflow.

Development workflow

Principles

> GRANDMA `workflow` is based on the Vincent Driessen’s branching model and a git-flow library of git subcommands to automate some parts of the flow making work more easily. However, GRANDMA development workflow is mainly concerned with a small part of the full git-flow process and essentially uses the *feature branch merged into develop one* management.



GRANDMA development workflow will use the following branch structure in local repositories:

- **Production branch** (called ‘master’)

  This branch represents the latest released / deployed / in-production code base. Only updated by merging other branches (mainly 'develop' branch) into it. Only GRANDMA mainteners can merged in it.

- **Development branch** (called ‘develop’)

  This is the main development branch where all the changes destined for the next release are placed, by merging feature branches into this branch.

- **Feature branches** (always prefixed with ‘feature/’)

  When you start work on anything non-trivial or even trivial, you create a feature branch. When finished, you’ll can publish your feature branch through a *Merge Request* as detailed in `pubworkflow` .

Only **Production** and **Development** branches should be visible in GRANDMA reference GitLab repository.

Developing a new feature

You interact with the GRANDMA repository by creating feature (or issue) branches in your local repository (cloned repository) and by submitting Merge Request to the GRANDMA reference repository.

First, create a feature branch. It's good practice to link all feature branches with an issue and to name feature branches descriptively. Here, issue is Issue #1; let's create a 'hello-world':

```
$> git flow feature start "Issue-#1-create-hello-world"
```

This produces output like:

```
Switched to a new branch 'feature/Issue-#1-create-hello-world'

Summary of actions:
- A new branch 'feature/Issue-#1-create-hello-world' was created, based on 'develop'
- You are now on branch 'feature/Issue-#1-create-hello-world'

Now, start committing on your feature. When done, use:

     git flow feature finish Issue-#1-create-hello-world
```

Doing the work

You can always verify you are in the proper feature branch:

```
$> git flow feature
```

Next, you create the hello world file for your feature request and commit the changes just as you would normally do.

```
$> touch hello-world.cc
$> … edition …
$> git status
$> git add hello-world.cc
$> … edition …
$> git add hello-world.cc
$> git commit -m 'Implement an awesome Hello World - Fixed #1'
```

> It's common to make multiple commits during the course of solving a problem and the work may go on for an extended period of time, maybe hours or even days. In that case it may be a good practice to clean up its local history and the commit messages. You can do this before finishing by rebasing as explained in `gitadvanced`.

Preparing the push

You may have to be sure that you are adding new code in an up-to-date*develop* branch; if needed you want to incorporate the recent changes from *origin* and possibly resolve early conflicts. So you collect the latest potential changes of the *origin* repository:

```
$> git fetch origin
$> git flow feature rebase 'Issue-#1-create-hello-world'
```

In your environment, there would be no conflicts … You can finish the feature:

Once the work is done and the feature branch merged into the*develop* one, you may publish it so that it could be added in reference repository after code reviews.

Publishing a feature branch

Check your work:

```
$> git flow feature diff 'Issue-#1-create-hello-world'
```

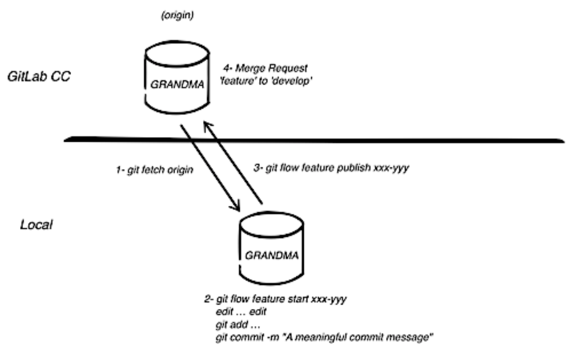
Then you publish your *feature* branch into the *origin* repository.

```
$> git flow feature publish 'Issue-#1-create-hello-world'
```

The next step details how to create a Merge Request to open a code review or asking a question or help.

Publishing workflows

Publishing a feature branch


(origin)
GitLab CC
GRANDMA
4- Merge Request 'feature' to 'develop'
1- git fetch origin
3- git flow feature publish xxx-yyy
Local
GRANDMA
2- git flow feature start xxx-yyy
edit … edit
git add …
git commit -m "A meaningful commit message"

With git-flow, pull requests from feature branches are always made against the 'develop' branch. Either way you'll need to push your changes so the merge master can pull them. To do so with git-flow we 'publish' the branch.

Once your changes are ready for a review or a merge request, you'll need to push them to your fork repository.

```
$> git flow feature publish 'Issue-#1-create-hello-world'

Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 390 bytes | 390.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote:
remote: To create a merge request for feature/Issue-#1-create-hello-world, visit:
remote:   https://gitlab.in2p3.fr/<your-id>/grandma/merge_requests/new?merge_request%5Bsource_branch%5D=feature%2FIssue-%231-create-hello-world
remote:
To gitlab.in2p3.fr:<your-id>/GRANDMA.git
 * [new branch]      feature/Issue-#1-create-hello-world -> feature/Issue-#1-create-hello-world
Branch 'feature/Issue-#1-create-hello-world' set up to track remote branch 'feature/Issue-#1-create-hello-world' from 'origin'.
Already on 'feature/Issue-#1-create-hello-world'
Your branch is up to date with 'origin/feature/Issue-#1-create-hello-world'.

Summary of actions:
- The remote branch 'feature/Issue-#1-create-hello-world' was created or updated
- The local branch 'feature/Issue-#1-create-hello-world' was configured to track the remote branch
- You are now on branch 'feature/Issue-#1-create-hello-world'
```

You may want to publish your ongoing*feature* branch before finishing it. Typically if you want to discuss some difficulties or choices in a Work In Progress [WIP] Merge Request …

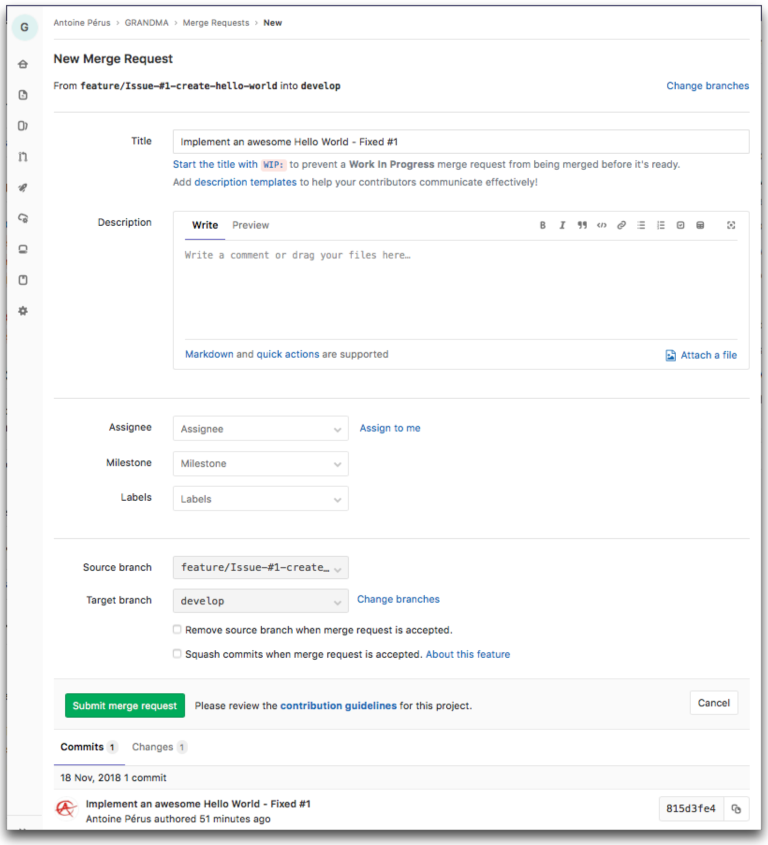
Creating a Merge Request

TL;DR:

1. Go to https://gitlab.in2p3.fr/GRANDMA/GRANDMA
2. Press `Create Merge Request` button, on the right.
3. Fill in an exhaustive description.
4. Verify that the selected source branch: `GRANDMA/GRANDMA` is the branch containing your changes `feature/Issue-#1-create-hello-world`.

   Verify that the target branch is the `develop` branch.

5. Ask for "Remove source branch when merge request is accepted."
6. Press `Submit Merge Request` button.


Antoine Pérus › GRANDMA › Merge Requests › New

New Merge Request

From `feature/Issue-#1-create-hello-world` into `develop` Change branches

Title: Implement an awesome Hello World - Fixed #1

Start the title with `WIP:` to prevent a **Work In Progress** merge request from being merged before it's ready.

Add description templates to help your contributors communicate effectively!

Description: Write | Preview

Write a comment or drag your files here…

Markdown and quick actions are supported. Attach a file

Assignee: Assignee — Assign to me

Milestone: Milestone

Labels: Labels

Source branch: feature/Issue-#1-create…

Target branch: develop — Change branches

☐ Remove source branch when merge request is accepted.

☐ Squash commits when merge request is accepted. About this feature

Submit merge request — Please review the **contribution guidelines** for this project. — Cancel

Commits 1 — Changes 1

18 Nov, 2018 1 commit

Implement an awesome Hello World - Fixed #1
Antoine Pérus authored 51 minutes ago — 815d3fe4

Details:

1. Go to https://gitlab.in2p3.fr/GRANDMA/GRANDMA

You will see at top of the display:

You pushed to feature/Issue-#1-create-hello-world at GRANDMA / GRANDMA 1 minute ago

1. Press `Create Merge Request` button, on the right.

You will arrive on a "New Merge Request" page.

1. Fill in an exhaustive description.

   You can change the title and fill in the description area in order to describe at best the work done with this MR.

   Choose too one or more "Labels" to help the MR categorization and list.

   You can select a reviewer in the "Assignee" pop-up.

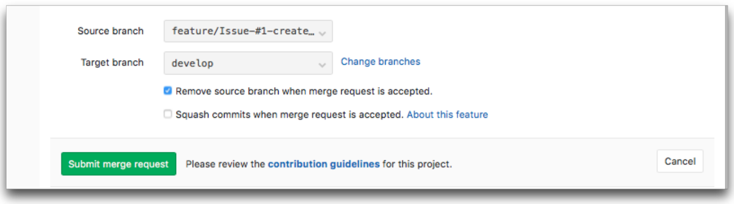
2. Verify the source branch and the target branch.

   The target branch should always be the `develop` branch. You can correct, il needed, clicking on `Change branches` .

3. Ask for "Remove source branch when merge request is accepted."

4. Press `Submit Merge Request` button.

   Check your commits and changes, then submit.


Source branch: feature/Issue-#1-create…

Target branch: develop — Change branches

☑ Remove source branch when merge request is accepted.

☐ Squash commits when merge request is accepted. About this feature

Submit merge request — Please review the **contribution guidelines** for this project. — Cancel

Updating the feature branch

Once the Merge Request opened, you may need to update your development in order to take remarks or requests into account. So you can resume your work session, continue the coding session and commit:

```
$> git flow feature checkout 'Issue-#1-create-hello-world'
$> … edition …
$> git add hello-world.cc
$> git commit -m 'Add some awesome variants - Fixed #1'
```

And eventually publish your work in order to inform the discussion:

```
$> git flow feature publish 'Issue-#1-create-hello-world'
```

If you go to your Merge Request on https://gitlab.in2p3.fr/GRANDMA/GRANDMA/merge_requests/<your-mr-number>, you will see your last commits included in the Merge Request.

Syncing with upstream

Scenario:

- You forked the GRANDMA repository some time ago.
- Time passes.
- There have been new commits made in upstream GRANDMA repository.
- Your forked GRANDMA repository is no longer up to date.
- You now want to update your forked GRANDMA repository to be the same as upstream.

Solution:

```
$> git checkout develop
$> git pull --rebase origin develop
$> git flow feature checkout 'Issue-#1-create-hello-world'
$> git flow feature rebase
$> git flow feature publish
```

The `--rebase` is only needed if you have local changes to the `develop` branch.

Deployment workflow

This documentation part is only for mainteners … To be written …

Merging code into master

Adding a tag version

Git essentials

Creating and switching branches

Note

> Never commit directly to the `master` branch. Avoid to commit directly to the `develop` branch.

Create a new branch and switch to it:

```
# creates a new branch off 'feature/current-feature' and switch to it
$> git checkout -b <branch-name> feature/current-feature
```

This is equivalent to:

```
# create a new branch off 'feature/current-feature', without checking it out
$> git branch <branch-name> feature/current-feature
# check out the branch
$> git checkout <branch-name>
```

Or using gitflow extension:

```
$> git flow feature <branch-name> feature/current-feature
```

To find out which branch you are in now:

```
$> git branch
```

Or to list existing feature branches:

```
$> git flow feature list
```

The current branch will have an asterisk next to the branch name. Note, this will list all of your local branches.

To list all the branches, including the remote branches:

```
$> git branch -a
```

To switch to a different branch:

```
$> git checkout <another-branch-name>
```

Delete local branch

To delete branch that you no longer need:

```
$> git branch -D <branch-name>
```

Staging and committing files

1. To show the current changes:

   ```
   $> git status
   ```

> To see everything unstaged diffed to the last commit:
>
> ```
> $> git diff
> ```

1. To stage the files to be included in your commit:

   ```
   $> git add path/to/file1 path/to/file2 path/to/file3
   ```

> To see everything staged diffed to the last commit:
>
> ```
> $> git diff --cached
> ```
>
> To see everything unstaged and staged diffed to the last commit:
>
> ```
> $> git diff HEAD
> ```

1. To commit the files that have been staged (done in step 2):

   ```
   $> git commit -m "This is the commit message."
   ```

Reverting changes

To revert changes to a file that has not been committed yet:

```
$> git checkout path/to/file
```

If the change has been committed, and now you want to reset it to whatever the origin is at:

```
$> git reset --hard HEAD
```

Stashing changes

To stash away changes that are not ready to be committed yet:

```
$> git stash
```

To re-apply last stashed change:

```
$> git stash pop
```

Advanced Git

Interactive rebase

Rebasing can be used to edit a commit message or to consolidate a collection of commits into a single commit. This is done by 'squashing' all non-desired commits.

```
$> git flow feature rebase -i 'Issue-#1-long-coding-session'
Will try to rebase 'Issue-#1-long-coding-session' which is based on 'develop'..
```

The interactive rebase (-i) edition will look something similar to:

pick 11c45c2 Reformulate pick 5e56ccf Add index on workflow pick d549e59 Clean up wrong sentence pick 2bd1c51 Improve workflow introduction pick 2f550fa Complete finishing feature section

```
# Rebase be16b12..2f550fa onto be16b12 (5 commands)
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
# d, drop = remove commit
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

Let say we want to improve the first commit message and merge the second with the forth. We will edit the rebase this way:

```
reword 11c45c2 Reformulate
pick d549e59 Clean up wrong sentence
pick 5e56ccf Add index on workflow
squash 2bd1c51 Improve workflow introduction
pick 2f550fa Complete finishing feature section

# Rebase be16b12..2f550fa onto be16b12 (5 commands)
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
# d, drop = remove commit
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

When we're done, we save and exit. If we look at our git-log output, it should have condense our feature branch into a simpler, more logical and descriptive one.