GIT SCM practices

Daniele Bacarella
Kiril Goguev

The following is a document that notes how the Erlang backend group will use git when developing the Erlang NetInf Backend.

Contents

Chapter 1

GIT Repository structure and policies

There will be 4 persistent branches.

- master

- staging

- develop

- release

The following naming convention for temporary branches is adopted:

- SprintX.shortStoryName

NOTE: ALL story tasks are done in the pair programming paradigm. The temporary branches will be deleted after each successful merge to the DEVELOP branch.

1.0.1 Policies

- master

  The 'master' branch will only contain Demo code. This is the code from the backend group which contains ONLY the fully tested and integrated stories.
  Tags will be made here under the following convention:
  SprintX.shortStoryName

This is a JENKINS build tool controlled area - No human user should be operating in this branch.

JENKINS is responsible for merging from 'master' to 'develop' at the end of a sprint- in order to keep the branches synchronized and provide a fresh clean start for each sprint from working demo code.

- release

  The 'release' branch will only contain individual stories which are completed and fully unit tested. Here we can pick and choose which stories to include in a specific demo. This branch is also a JENKINS build tool area.

  JENKINS is responsible for integration testing and merging between 'release' and 'master'

- develop

  The 'develop' branch will contain all the code this is able to be compiled on the server and is where the human users will start their personal story branches. Also a JENKINS build tool area, The code here will be considered in a "Story done and compiles but not yet tested" state.

  JENKINS is responsible for unit testing and merging between 'develop' and 'release'

- staging

  The 'staging' branch will contain all the dirty code and is where the human users will push all their code when finished for the end of the day. Also a JENKINS Build tool area, the code here will be considered in a story is in progress it may be done but it also may not compile. JENKINS will pull all the code from this branch and try to compile it, if it compiles then it will be merged with the 'develop' branch.

- SprintX.shortStoryName

  The branch's name will contain the local working code for the specific sprint story followed by a short story name-typically the name written on the post it note for example: MSG_Handler. A merge to the DEVLEOP branch will mean the story is considered done for the

4

sprint but requires testing by integration tools and JENKINS. This branch will be deleted after the tests are passed and the a successful merge is complete. History will be kept in the 'develop' branch should we need to revert for any reason.

Chapter 2

Naming conventions

Where ever possible, the JENKINS build tool will be configured to create the branches and tags accordingly.

2.0.2 Branches

With the exception of the RELEASE and the MASTER, each branch will be labelled as follows: SprintX.StoryY -where X is the sprint number and Y is the Story number

2.0.3 Tags

There will be at least 2 places in the GIT repository where we will use tags.

1. master

2. release

Each tag will be labelled as follows SprintX.shortStoryName-RELEASE or SprintX-DemoZ depending on the sprint and which branch is being pushed to.

Chapter 3

Sample work flow

The following section will detail a sample work flow for a sprint with at least two story items.

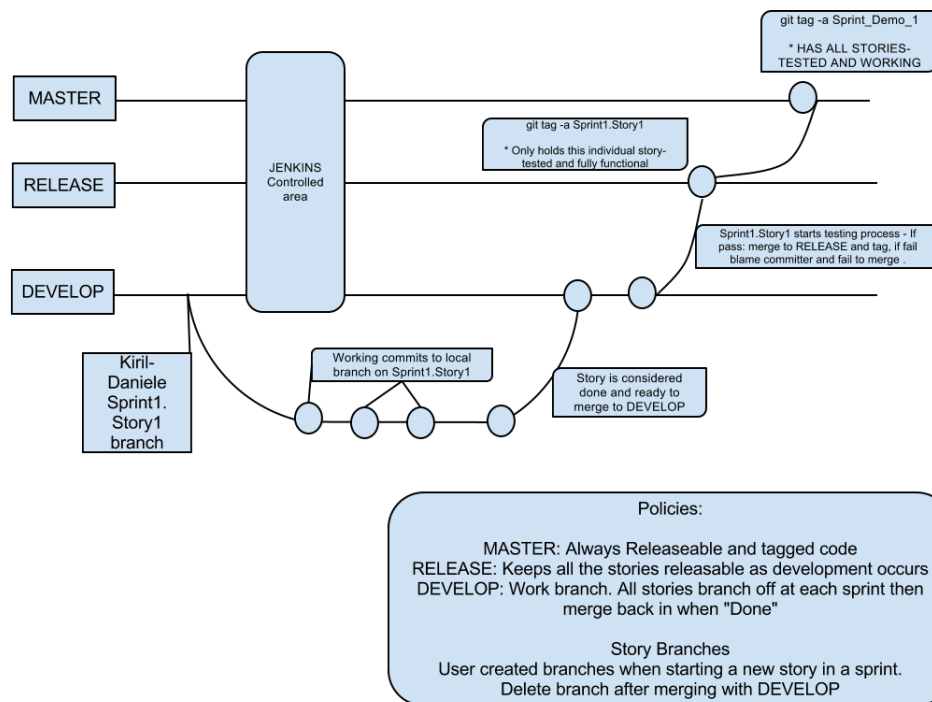The follow image can quickly show you how the individual story may look on a time-line.

Figure 3.1: Sample workflow for one story item

3.0.4 Starting a sprint

At the start of each sprint the development team will perform a git checkout
-b SprintX.shortStoryName DEVELOP
    for a running example we will take the HTTP Handler from the first
sprint.

```
git checkout -b SprintX.shortStoryName develop
git push -u origin SprintX.shortStoryName
```

This will pull all the previous items into your temporary working branch
from develop

3.0.5 Working during a sprint

As you and your partner work together to complete the tasks outlined in
the story, commit as usual to your local branch.

```
git pull origin staging
git add [files that have been added or created/modified]
git commit -m description - where description is the commit message you would like as
```

3.0.6 Deploying a Done story

After all the tasks are completed and you feel ready to move the task to
the DONE state, merge it to the development branch using the following
command:

```
git checkout staging
git merge --no-ff SprintX.shortStoryName
git push origin staging


- do the following only if you wish to delete your branch on the remote repository -
git push origin :SprintX.shortStoryName

- do the following if you wish to delete your branch locally -
git branch -d SprintX.shortStoryName
```

This will perform a merge and fast-forward on the remote 'staging' branch while keeping historical information. This will also create empty commits in the development branch as a way to quickly spot revisions which you may wish to go back to, for example if you need to remove a story that was not working properly.

NOTE: AS A HUMAN USER THIS IS TYPICALLY WHERE YOU STOP WORKING IN THE WORKFLOW.
THE FOLLOWING STEPS ARE TO BE IMPLEMENTED BY JENKINS AND AUTOMATED BUILD TOOLS. YOU ARE ONLY CONCERNED WITH WORKING ON THE DEVELOP BRANCH AND YOUR OWN STORY BRANCHES FOR EACH SPRINT. YOU SHOULD NEVER TOUCH MASTER OR RELEASE UNLESS ABSOLUTELY NECESSARY

3.0.7 Testing a sprint story

The following steps assume that a testing suite was created and works well with JENKINS build tool.

3.0.8 Merging in to develop

As a story branch has been merged to 'develop', the code goes through a set of unit tests and when it successfully passes all of them, it is ready to be merged to 'release':

```
git checkout develop
git merge --no-ff staging
git push origin develop
```

3.0.9 Merging in to release

As a story branch has been merged to 'develop', the code goes through a set of unit tests and when it successfully passes all of them, it is ready to be merged to 'release':

```
git checkout release
git merge --no-ff develop
git tag -a SprintX.shortStoryName-RELEASE
git push origin release
```

3.0.10 Merging in to master

Supposing that all the sprint stories code have reached the stage of release, before proceeding with the final merging , we check that all the desired features have been implemented and tested properly. The final merging is performed upon the MASTER branch containing always the stable version of all the code written so far.

```
git checkout master
git merge --no-ff release
git tag -a Sprint1.Demo1
git push origin master
```

3.0.11 Post-sprint practice

After each sprint we should synchronize the 'master' and the 'develop' branch. This ensures a clean start for each sprint as the 'staging' branch will inherently be considered the "dirty" branch.

```
git checkout DEVELOP
git merge --no-ff MASTER
```

Chapter 4

References

The following was an inspiration for the git model:

http://nvie.com/posts/a-successful-git-branching-model/