Git mini-tutorial

28 May. 2012

Benoit RAT

Contents

In git you should always work on your own branch.

- The first time you need to create it.
- Or if someone create you the branch you can only use it.

1 First time (create your own branch)

Download the repository.

```
git clone git@ohwr.org:white-rabbit/wr-nic.git
```

Create our own branch:

```
git branch sevensols
```

Switch to our own branch:

```
git checkout sevensols
```

Create an empty test file:

```
touch TEST
```

See the status of the repository:

```
git status
```

Add a file that you have created:

```
git add TEST
```

then commit it to your local branch:

```
git commit -m "blablabla"
```

Finally commit your local branch to the remote repository (**FOR THE FIRST TIME**), and if you have the permission to do it.

```
git push origin sevensols
```

2 First time (change to a created branch)

Download the repository

```
git clone git@ohwr.org:white-rabbit/wr-nic.git
```

Switch to your created branch

```
git checkout sevensols
```

Switch to your created branch

```
git checkout sevensols
```

Check that you are on the good branch

```
git branch
```

You should see something like this:

```
master
* sevensols
```

Which means that sevensols is the actual branch.

3 Committing

In git your commit when you have done one action, e.g, correct a bug, add a feature, add a module. In other words, each commit should represent an independant and usable stage in your project.

3.1 Check the status

You should always check the status of your branch before and after committing

```
git status
```

3.2 Adding your files

for example we have created in the folder *"./modules/wr_dio/"* the two following files:

- wr_dio.wb
- wr_dio.vhd

you can add them doing:

```
git add modules/wr_dio/wr_dio.wb
git add modules/wr_dio/wr_dio.wb
```

Or by doing (This will add all the files the directory "./modules/wr_dio/")

```
git add modules/wr_dio/.
```

Or by doing: (This will add all the files in a recursive way from the current directory)

```
git add .
```

3.3 Comitting the changes

You should describe the changes done to ease the understanding of waht you have done in the code for other people

```
git commit -m "Adding the dio module in the repository"
```

And then you can send your change to remote repository

```
git push
```

4 Updating the master branch

you should first switch back to the master branch

```
git checkout master
```

Maybe you can verify on which branch you actually are

```
git branch
```

And then download the change by doing a pulling

```
git pull
```

5 Undoing

A good tutorial can be found here http://learn.github.com/p/undoing.html

5.1 Unstage

If you want to remove a file from a commit (unstage it) you can use

```
git reset HEAD <file>
```

You can also run a good tools to do this called `git-cola`

5.2 Unmodifying

WARNING: If you do this you will lost the *changes*:

```
git checkout -- <file>
```

Or if you have a lot of files:

```
git reset --hard HEAD
```

6 Git ignore files

To avoid commiting file that you don't want to track you should add them to the *.gitignore* file. You can use the wildcard to ignore some files, here is an example of root *.gitignore*

```
# .gitignore file
# OS generated files #
######################
*.bak
*.*\#
\#*
.\#*
```

```
*.*~
*.swp

# Compiled source #
###################
*.bit
*.exe
*.o
*.so
*.wlf
*.vstf
*.vcd
```

Then each subfolder may also have its own .gitignore for example we have for:

- ip_cores/.gitignore:

```
# .gitignore file (for ip_cores/)
# We do not want to track anything in this directory because
# it is where modules are fetch from other repository
*
```

- syn/spec/.gitignore: Here we only want the manifest and maybe some config files but not all the generated file for ISE synthesis

```
# .gitignore file (for syn/*/)
# Folder where the synthesis is done
# We want to ignore all the file generated by ISE for synthesis
# except the Manifest.py and the xise project
*
!Manifest.py
!wr_nic.xise
```

7 Syncronization Master & Branches

7.1 Master => branch

Making sure changes on master appear in your branch

```
git rebase master
```

7.2 branch => master

This one is more delicate

First, switch back to the master branch:

```
git co master
```

Check to see what changes you're about to merge together, compare the two branches:

```
git diff master xyz
```

If you're in a branch that's not the xyz branch and want to merge the xyz branch into it:

```
git merge xyz
```

7.2.1 Reverting changes to before said merge

```
git reset --hard ORIGHEAD
```

8 Using Patch

You can create a list of patch by doing:

```
git format-patch <revision>
```

Then you can apply them using

```
git am <0001-nombre-del-patch>
git am <0002-nombre-del-patch>
```

If you obtain a conflict for the second patch you can solved it by searching the files that are in conflict:

```
git status
```

And resolving it by hand using meld (or similar)

```
meld <conflict files>
```

Once you have

```
git add  <the fixed files>
git am --resolved
```

9 Rebasing (Ordering and removing errors)

If you want to keep as clean as possible a list of patch, you can use the rebase command (see [rebase]):

First you obtain the log of your patches:

```
git log --oneline
```

```
f1500fc bopyt: bad commit msg
60fa4ab ddr: patch final
bf44f22 test: Stupid test that can be removed from history
536741f ddr: Step in between (not usufull)
b611a95 cpu: Something we want to keep
c7d4d29 ddr: first intent to use ddr
```

```
be6e661 boot: something about boot which is correct
...
```

Then you might save the state using a tag in case something went wrong

```
 git tag "v3.1-notrebased"
```

Then you need to rebase: for this you need to select the step before the thing you want to change:

```
git rebase -i be6e661
```

And you will have to modify the file like:

```
## Original file
pick c7d4d29 ddr: first intent to use ddr
pick b611a95 cpu: Something we want to keep
pick 536741f ddr: Step in between (not usufull)
pick bf44f22 test: Stupid test that can be removed from history
pick 60fa4ab ddr: patch final
pick f1500fc bopyt: bad commit msg
```

```
## Saved file
pick b611a95 cpu: Something we want to keep
pick c7d4d29 ddr: first intent to use ddr
fixup 536741f ddr: Step in between (not usufull)
squash 60fa4ab ddr: patch final
reword f1500fc bopyt: bad commit msg
```

And you will obtain something like

```
8ac18d8 boot: bad commit msg corrected
ac85ae5 ddr: patch final
c554a3e cpu: Something we want to keep
be6e661 boot: something about boot which is correct
```

In case you have done something bad you can go back to your tag[1] (by doing an hard reset)

```
git reset --hard v3.1-notrebased
```

10 Misc

10.1 Dropbox

If you want to use git and dropbox at the same time you can mount your dropbox directory somewhere in your git path

```
sudo mount --bind SOURCEDIRECTORY TARGETDIRECTORY
```

11 References

- Git cheat-sheet: http://help.github.com/git-cheat-sheets/

[1] In the case you don't have a tag, you can find the hash previous the rebase using: `git reflog --all`

- Learn Git: http://learn.github.com/
- Git SVN crash course: http://git.or.cz/course/svn.html
- Ignore files: http://help.github.com/ignore-files/
- Rebasing: http://codeutopia.net/blog/2009/12/10/git-interactive-rebase-tips/

This document is written in markdown syntax and can be generated using pandoc