

**GIT**

**Commands and best practices**

**v0.4**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Redactor | Modification |
| 0.1 | 02/10/2012 | Jérôme COUDERT | Création |
| 0.2 | 04/10/2012 | Thomas DIAVET | Update |
| 0.3 | 12/10/2012 | Vincent LEPOT | Update |
| 0.4 | 19/11/2012 | Jérôme COUDERT | Update |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Summary**

[1 Installing GIT 3](#_Toc341097834)

[2 Branch creation 4](#_Toc341097835)

[2.1 Create a new branch 4](#_Toc341097836)

[2.2 Retrieve an existing project branch from the origin 4](#_Toc341097837)

[3 Updating branches 5](#_Toc341097838)

[3.1 Updating a branch project from origin (Retrieve the modifications of other people) 5](#_Toc341097839)

[3.2 Updating a branch from master 5](#_Toc341097840)

[3.3 Merging the branch project into the release branch 5](#_Toc341097841)

[4 Deleting branches 6](#_Toc341097842)

[4.1 Deleting a branch from its local repository 6](#_Toc341097843)

[4.2 Deleting a branch from origin 6](#_Toc341097844)

[5 Other commands 6](#_Toc341097845)

[5.1 List all remote branches 6](#_Toc341097846)

[5.2 List all local branches 6](#_Toc341097847)

[5.3 List all local and remote branches 6](#_Toc341097848)

[5.4 Switch to another branch 6](#_Toc341097849)

[5.5 Branch status 7](#_Toc341097850)

[5.6 Add file 7](#_Toc341097851)

[5.7 Add directory 7](#_Toc341097852)

[5.8 Save changes 7](#_Toc341097853)

[5.9 Listing last commits of the current branch 7](#_Toc341097854)

[5.10 Listing last commits of another local branch 7](#_Toc341097855)

[5.11 Listing last commits of a branch on origin 7](#_Toc341097856)

[5.12 Searching a branch 7](#_Toc341097857)

[5.13 Listing modified files of a branch compared to the prod 8](#_Toc341097858)

[5.14 Reverting to the last commit 8](#_Toc341097859)

[5.15 Aborting a merge 8](#_Toc341097860)

[5.16 Merging by ignoring conflicts due to changes of spacing (can facilitate some merges) 8](#_Toc341097861)

[5.17 Resolving conflicts 9](#_Toc341097862)

[6 Best pratices 9](#_Toc341097863)

[7 Tools 10](#_Toc341097864)

[8 To learn more 10](#_Toc341097865)

# Installing GIT

1. Create a JIRA ticket for the Exploitation in order to request the creation of a DevNG account
2. Download and install [GIT](http://git-scm.com/downloads)
3. GIT includes a consol (Git Bash) which allows executing commands. To do this, you can also install [Putty.exe](http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html)  
   The advantage of using Putty is to be connected directly to the remote repository by ssh. You don’t need to synchronize your local repository with the remote repository. Commands are faster when they are executed on the filer server
4. In Windows Explorer, connect a network drive on [\\share.dating.dev\dating](file://share.dating.dev/dating)  
   To do this, use the information received in the confirmation email account. You will be able to access to the DevNG files on your computer.
5. Clone the remote repository to retrieve the source code

Cloning the repository

Solution 1: Using PuTTY on share.dating.dev  by ssh

$ git clone ssh://username@git.dating.dev/mcorp/data/git/dating.git $HOME/mcorp/www/meetic/.

Solution 2: Using GIT Bash to clone on the network drive (**much longer**)

$ git clone ssh://username@git.dating.dev/mcorp/data/git/dating.git Z:/meetic/.

*After this action, a folder will be created with all the pages and source codes (PHP, JS, CSS, Log, Img)*

From this moment, only the Master branch (identical to the prod) will be created and be visible on its local repository. It is necessary to synchronize its repository by retrieving all branches from the origin.

Synchronize

$ git fetch –p

*The parameter-p allows deleting the obsolete branches. This command must be executed at least once a day.*

Now, it’s possible to create new branches.

# Branch creation

## **Create a new branch**

Update its local repository relating to origin   
$ git fetch origin  
  
Create a new branch from the master branch   
$ git checkout –b <branch\_name> origin/master

Push the branch to the origin  
$ git push –u origin <branch\_name>

*The parameter –u specifies to GIT that your branch has to track the branch pushed to the origin*

|  |  |  |
| --- | --- | --- |
| Branch type | Branch naming rule | Comment |
| Project | project/<product>-<ProjectNumber>-<Project\_Title>-<owner>  project/Dating-GDO-2258-D6\_Push\_To\_Activity-MF1 project/Dating-M-35223-wording\_error-jcoudert | **<product>** - Dating - Matchmaking **<ProjectNumber>**  - the JIRA sent to QA (GDO-XXXX) - the Mantis (M-XXXX) - the IOP JIRA (IOP-XXXX) **<Owner>**  - username of the owner (eg: vlepot)  - the scrumteam code (eg: MF1) |
| Hotfix | hotfix/<product>-<MantisNumber>-<Title>-<owner>  hotfix/Dating-M-35223-wording\_error-jcoudert | **<product>** - Dating - Matchmaking **<MantisNumber>**  - the Mantis (M-XXXX) **<Title>**  - Title of the mantis **<Owner>**  - username of the owner (eg: vlepot)  - the scrumteam code (eg: MF1) |

## **Retrieve an existing project branch from the origin**

Update its local repository relating to origin  
$ git fetch origin  
  
Pull and switch on the branch   
$ git checkout –b <branch\_name> origin/<branch\_name>  
*<branch\_name>is the name of the branch already on origin*

# Updating branches

## **Updating a branch project from origin** (Retrieve the modifications of other people)

Switch to **your branch to be updated**

$ git checkout <branch\_name>  
  
Merge the modifications into your branch

$ git fetch

$ git merge origin/<branch\_name>

*It’s better to execute these 2 commands than a simple git pull in order to update the index*

*Fix possible merge conflicts*

Push the updated branch to origin

$ git push origin <branch\_name>

## **Updating a branch from master**

This command must be executed at least once a day (by only one member of the team if several people are working on the same branch)

Switch to **the branch to be updated**

$ git checkout <branch\_name>

Retrieve the last version of prod

$ git fetch

Merge your branch into the master

$ git merge origin/master

*Fix possible merge conflicts*

Push the updated branch to origin

$ git push origin <branch\_name>

## **Merging the branch project into the release branch**

Switch to the **project branch**

$ git checkout <project\_branch\_name>

Retrieve the last version of prod

$ git pull origin master

*Fix possible merge conflicts*

Push the updated branch to origin (for later traceability if needed)

$ git push origin <project\_branch\_name>

Pull the release branch and switch it on

$ git checkout -b <release\_branch\_name> origin/<release\_branch\_name>

Merge the project branch into the release branch

$ git merge <project\_branch\_name>

*Fix possible merge conflicts*

Push the updated release branch to origin

$ git push origin <release\_branch\_name>

# Deleting branches

## **Deleting a branch from its local repository**

Switch **to another branch** than the one you want to delete   
$ git checkout <another\_branch\_name>  
  
Delete the branch

$ git branch –D <branch\_name\_to\_delete>

## **Deleting a branch from origin**

This command must be executed systematically after MEP

Remove branch from origin

$ git push origin :<branch\_name\_to\_delete>

# Other commands

## **List all remote branches**

$ git branch –r

## **List all local branches**

$ git branch

*The active branch is marked by \**

## **List all local and remote branches**

$ git branch -a

## **Switch to another branch**

$ git checkout <branch\_name\_to\_switch>

*Make the current branch <branch\_name\_to\_switch>, updating the working directory to reflect the version referenced by <branch\_name\_to\_switch>*

## **Branch status**

List files added to the working directory, files with changes, and untracked (ie. What will be saved to the next commit)

$ git status

## **Add file**

Add files to the branch which will be committed later

$ git add <file\_path>

## **Add directory**

Add all files under the directory <directory\_name> to the branch, including subdirectories

$ git add <directory\_name>

## **Save changes**

**Solution 1**  
Add files to the branch.

$ git add <file\_path\_1> <file\_path\_2>

Commit changes and add a comment

$ git commit -m ‘My changes are…’  
*-m allows adding a comment*

**Solution 2**

Add and commit in one command

$ git commit -am ‘My changes are…’

*Solution 2 is to be avoided. The first one allows listing all files before adding a comment and having a better visibility regarding which changes will be committed.*

## **Listing last commits of the current branch**

$ git log

## **Listing last commits of another local branch**

$ git log <branch\_name>

## **Listing last commits of a branch on origin**

$ git log origin/<branch\_name>

## **Searching a branch**

Search branches containing a specific word

$ git branch –r | grep <word\_to\_search>

## **Listing modified files of a branch compared to the prod**

Update your branch and merge it into the master branch

$ git fetch

$ git merge origin/master

*Fix possible merge conflicts*

List files modified on the branch (modified, added, deleted)

$ git diff –-name-only origin/master

List files modified on the branch and their status (A/M/D)

$ git diff –-name-status origin/master

View the changes on a modified file

$ git diff <file\_name> origin/master

Show all changes on the branch

$ git diff origin/master

## **Reverting to the last commit**

Abandon everything since your last commit  
$ git reset --hard

## 

## **Aborting a merge**

After a merge, if you have conflicts that are too difficult to resolve without the help of another person and you have another priority, you can abort it.  
$ git merge –abort

Never commit if the merge is not resolved

## Merging by ignoring conflicts due to changes of spacing (can facilitate some merges)

Use the command

$ git merge –Xignore-space-changes

Instead of

$ git merge <branch>

## Resolving conflicts

List files with conflicts

$ git status

*Files with conflicts are listed in the ”Unmerged paths” part*

For each conflicted file, solve conflicts marked by section starting with

<<<<<<< HEAD

Ending with

>>>>>>> <merged\_branch>

Separated by

=======

Resolving conflict is:

• Or choose between the 2 blocks of code

• Either restores the correct block of code from the 2 versions

Once ALL conflicts are resolved, mark it by adding file

$ git add <file>

Once ALL files with conflicts are resolved, commit changes without adding a comment (keep the suggested comment)

$ git commit

*(Without option)*

Push the branch to origin

$ git push origin <branch\_name>

To check that all conflicts are resolved

$ grep -E '^(<<<<|====|>>>>)' <file\_name>

*This command does not return anything*

Please, check that the files are syntactically correct

# Best pratices

* Update branches from the master several time a day
* Removed branches on origin after the project is launched
* Before merging, check impacted files not to add files inadvertently
* Carefully following the standards (4 spaces indentation, no tabs, file in UTF-8 without BOM, Unix line endings) can reduce the number of conflicts to resolve

# Tools

* WinMerge : software to manage conflicts
* gitK : allows to visualize the branches and commits graphically
* TortoiseGIT
* Git Bash : to execute GIT commands directly on your local repository
* Putty : to be connected by ssh (share.dating.dev) and execute GIT commands directly to the remote repository (faster)

# To learn more

* <http://git-scm.com/>
* https://github.com/
* <http://en.wikipedia.org/wiki/Git>
* <http://cheat.errtheblog.com/s/git>
* http://gitref.org/