

https://i.stack.imgur.com/zUInQ.png

* **git fetch** is the command that says "bring my local copy of the remote repository up to date."
* **git pull** says "bring the changes in the remote repository where I keep my own code."

Normally "**git pull**" does this by doing a "**git fetch**" to bring the local copy of the remote repository up to date, and then merging the changes into your own code repository and possibly your working copy.

In the simplest terms, git pull does a git fetch followed by a git merge.

|  |  |
| --- | --- |
|  |  |
| The take away is to keep in mind that there are often at least **three copies** of a project on your workstation. One copy is your own repository with your own commit history. The second copy is your working copy where you are editing and building. The third copy is your local "cached" copy of a remote repository. |  |

$ git config --global user.name "AmineLouati"

$ git config --global user.email louati.amiine@gmail.com

Il faut vérifier 2 choses :

1. Le nombre de commit est le même
2. L’identifiant du dernier commit est le même

Mettre à jour le repo local d’un remote fork par rapport au remote origin

0. se positionner sous le dossier du projet

1. git remote add upstream git://github.com/ow2-proactive/scheduling.git

2.1 git fetch upstream // Now you can collect the latest changes of the upstream repository with fetch

*2.2 git checkout master*

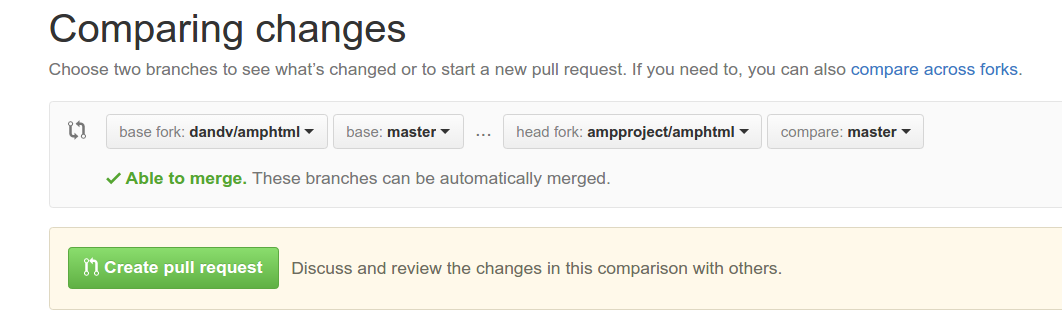
*2.3 git merge upstream/master*

3. git pull upstream master

4. git push origin master

5. ./gradlew spotlessApply clean build -x test  // faire le build sans les test après avoir récupéré une version de l’origine

Another method

1. Open your fork on GitHub.
2. Click on Pull Requests.
3. Click on New Pull Request. By default, GitHub will compare the original with your fork, and there shouldn't be anything to compare if you didn't make any changes.
4. Click switching the base if you see that link. Otherwise, manually set the base fork drop down to your fork, and the head fork to the upstream. Now GitHub will compare your fork with the original, and you should see all the latest changes. [](https://i.stack.imgur.com/FoNQu.png)
5. Create pull request and assign a predictable name to your pull request (e.g., Update from original).
6. Scroll down to Merge pull request, but don't click anything yet.

Créer un fichier gradle.properties dans le dossier caché .gradle du home (cd home) et ajouter la ligne suivante :

org.gradle.daemon=true 🡪 le build devient plus rapide

gradle gère le cycle de vie de tout le projet y compris la compilation, les tests …

La creation d’un nouveau sous module 🡪 l’ajouter dans le fichier settings.gradle

les builds se font toujours par ligne de commande.

Nous pouvons faire un build d’un sous module

faire un test sur une méthode

Junit tutorial fogella

Gradle tutorial

 Pour créer une branche et y basculer tout de suite, vous pouvez lancer la commande git checkout avec l'option -b :

$ git checkout -b prob53

Switched to a new branch "prob53"

C'est un raccourci pour :

$ git branch prob53

$ git checkout prob53

Vous pouvez l'effacer avec l'option -d de la commande git branch :

$ git branch -d correctif

Deleted branch correctif (3a0874c).

Vous pouvez lancer vos tests dans la branche correctif, vous assurer que la correction est efficace et la fusionner dans la branche master pour la déployer en production. Vous réalisez ceci au moyen de la commande git merge :

$ git checkout master

$ git merge correctif

## Git commit squash mode interactif

$ git rebase -i HEAD~5

## Au cas où ça marche pas, on peut revenir à l’état précédent

git rebase --abort

Pour annuler des commits en local, il existe la commande git reset.

Just use git reset without the --hard flag:

git reset HEAD~1

## Pour annuler un merge

git merge --abort

# [**How do I properly force a Git push?**](https://stackoverflow.com/questions/5509543/how-do-i-properly-force-a-git-push)

git push origin <your\_branch\_name> --force

<https://makina-corpus.com/blog/metier/archives/git-annuler-proprement-un-commit-apres-un-push>

These three commands have entirely different purposes. They are not even remotely similar.

## **git revert**

This command creates a new commit that undoes the changes from a previous commit. This command adds new history to the project (it doesn't modify existing history).

## **git checkout**

This command checks-out content from the repository and puts it in your work tree. It can also have other effects, depending on how the command was invoked. For instance, it can also change which branch you are currently working on. This command doesn't make any changes to the history.

## **git reset**

This command is a little more complicated. It actually does a couple of different things depending on how it is invoked. It modifies the index (the so-called "staging area"). Or it changes which commit a branch head is currently pointing at. This command may alter existing history (by changing the commit that a branch references).

## **Using these commands**

If a commit has been made somewhere in the project's history, and you later decide that the commit is wrong and should not have been done, then git revert is the tool for the job. It will undo the changes introduced by the bad commit, recording the "undo" in the history.

If you have modified a file in your working tree, but haven't committed the change, then you can use git checkout to checkout a fresh-from-repository copy of the file.

If you have made a commit, but haven't shared it with anyone else and you decide you don't want it, then you can use git reset to rewrite the history so that it looks as though you never made that commit.

## SSH bitbucket

<https://confluence.atlassian.com/bitbucket/set-up-ssh-for-git-728138079.html>