Some (hopefully) helpful tips for using git

Mateusz Krajewski

March 12, 2022

Contents

1 Introduction

It is of paramount importance that we use git in an organised and consistent manner. It will reduce confusion and make it easier to mitigate the consequences of our mistakes. And there will be plenty, inevitably. The following is my attempt to standardise our git workflow. Feel free to share this document and adjust it to Your needs – just remember to keep my name in. Contributions will be appreciated.

In order to make the branch history as clean as possible, we have decided to create seven separate branches - one for every team member. Every developer is going to work on their issues on the branch assigned to them - and merge it with main when necessary.

I am going to present a couple of common scenarios that can arise when working with git. I tried to make it as relatable to the current project as possible. For that reason, all the examples below are based solely in Eclipse IDE. I may include some command-line examples later on.

2 The development process

The following is the general overview of our development process. You should try to adhere to it as much as possible. To explore the steps in more details, refer to the later parts of this document.

1. Start working on the issue on Your own branch.
2. Remember to commit and push regularly, so that other team members can review Your changes.
3. Your work is only done when Your code reviewer says so.
4. In order to merge, first make sure that Your local main branch is up-to-date with the remote one. Switch to main and pull from the remote.
5. Then, merge Your branch into main. Solve any conflicts that may arise.
6. Close the issue in the issue tracker.

3 Miscellany

3.1 Git staging

There are two helpful views that You can enable in order to better understand what's going on. One of them is *Git staging*. You'll find it in:

`Window` → `Show view` → `Other` → `Git` → `Git Staging`.

It's an easy tool for reviewing the changes You have made, and ultimately committing them.

3.2 History

You can find *History* view under:

`Window` → `Show view` → `Other` → `Team` → `History`.

It shows the history of changes made to the chosen resource in the *Package Explorer*. Besides, it makes it extremely simple to see the state of current branches, both local and remote.

3.3 Current branch

The branch You're currently in is displayed right next to the project name in *Package Explorer* (Figure 1). You almost certainly know this. But if I want to make this document as comprehensive as possible, it needs to be here.

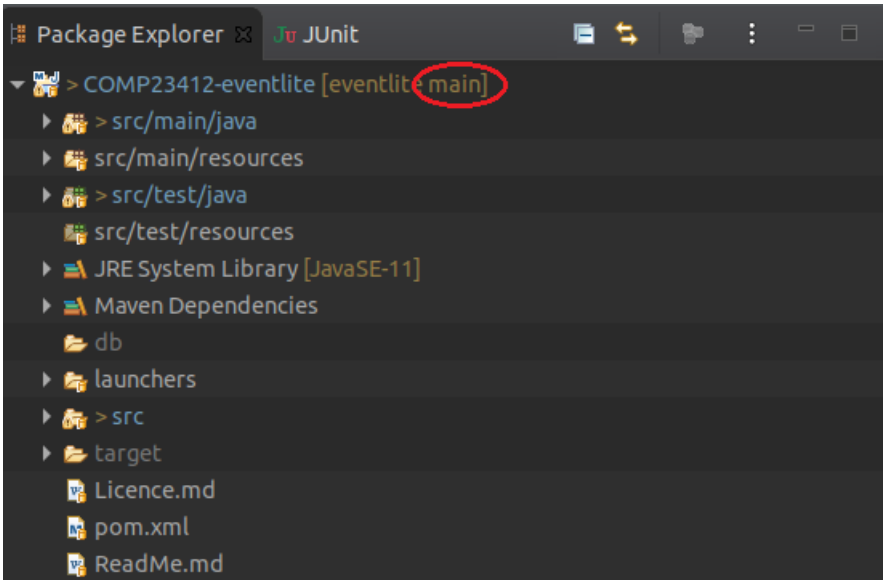

Figure 1: The location of the current branch name.

3.4 Pulling

In order to synchronise Your local branch with any changes made to its remote counterpart, right-click the project name and select:

`Team` → `Pull`.

Most likely, You will have to choose between two identical *Pull* options. The only difference is that the second one goes into the process in more detail. Ultimately, though, it doesn't really matter which one You'll pick.

4 Moving between branches

This is probably basic knowledge as well. In Eclipse, You can switch between branches/commits by right-clicking the project name in the *Package Explorer* and choosing:

`Team` → `Switch to`.

With the help of this tool, it is possible to:

- Create a new local branch;
- Create a local copy of a remote branch;
- Switch between local branches;
- Switch between commits.

NOTE: Before You do any of that, make sure to look into Git staging and commit any changes You have made so far. Otherwise, git may behave unpredictably and You will spend the next two hours trying to figure out what to do (it certainly never happened to me, duh).

5 Updating the feature branch

Suppose You started working on Your branch and already made some commits. In the meantime, however, one of Your teammates merged their work into main and now Your branch is outdated. In such a case, the commit graph may look (roughly) like in Figure 2. You may want to update Your branch and keep working on it, without having to merge into main (not yet, at least). In order to do that, You can do the opposite - merge main into Your branch. This operation would result in creating a new merge commit, just like in Figure 3.

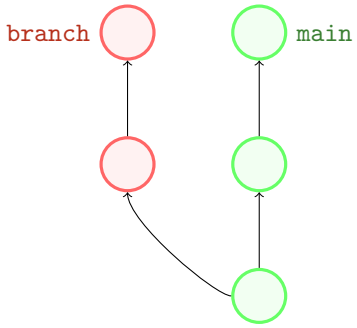

Figure 2: Before updating.

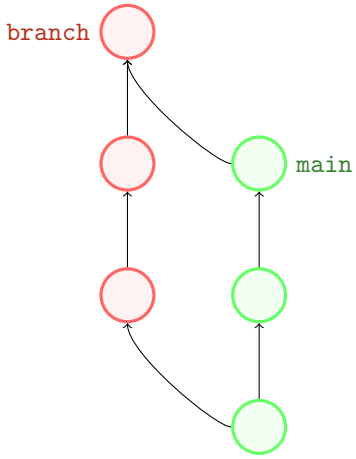

Figure 3: After updating.

Now, how to do that? The process itself is quite simple. First, make sure that the main branch is synchronised with the remote (switch to main and pull). Then, switch to the feature branch. Right-click the project name and choose the following:

`Team` → `Merge`.

After that, select the local main branch. Solve any conflicts that may arise. Move on.

6 Merging with main

Merging is really quite simple. Before You do that, however, make sure that Your main branch is up-to-date with the remote one (switch to main and pull the remote). After that, right-click the project name and choose:

`Team` → `Merge`.

Then, select the branch You want to merge into main. No need to meddle with the additional options. Now, if we imagine a commit graph like that in Figure 2, its structure after the merge would be similar to that in Figure 4. Note how git creates an additional merge commit, which incorporates the changes made both in the main branch and in the feature branch. Any further changes made to the feature branch will result in the graph evolving like in Figure 5.

If You encounter conflicts, refer to section 7.

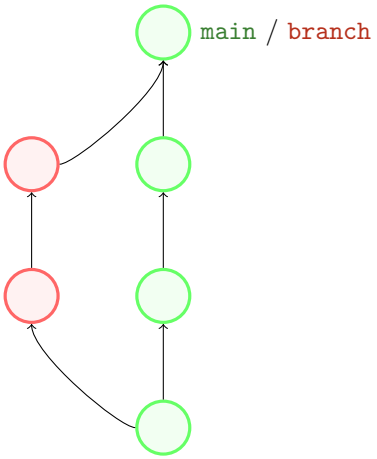

Figure 4: After merge.

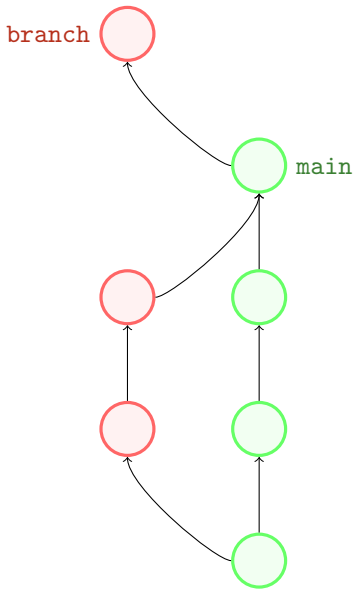

Figure 5: After further changes.

7 Solving conflicts

Merge conflicts are inevitable. No matter what You're trying to achieve by merging, they will often be there. And they will need to be solved. A conflict is signified by a window similar to that in figure 6.

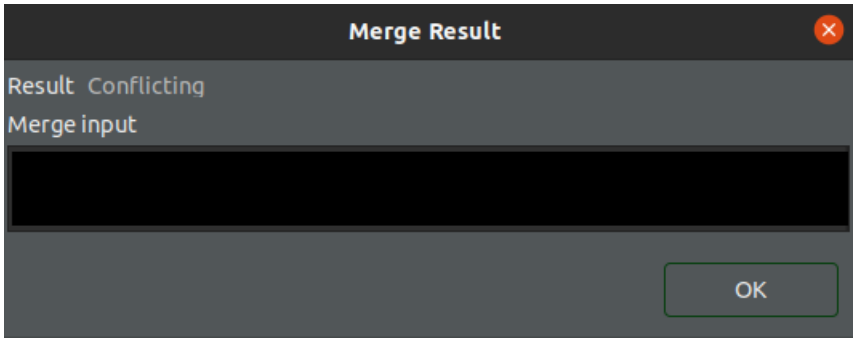

Figure 6: A conflict message.

The *Unstaged changes* section in Git Staging lists all files where conflicts were detected (Figure 7). Successfully merged files are in the *Staged changes* section. Now, every conflict can be solved in two ways - either manually or by using a *Merge Tool*.

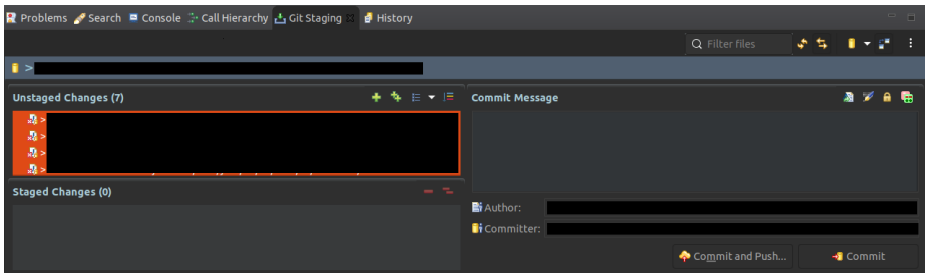

Figure 7: Git Staging view during a merge conflict.

7.1 Manually

Following the hints from Git Staging, find and open the respective files in the *Package Explorer*. Every conflict would be of the form:

```
<<<<<<< HEAD
what you see in the branch You're merging into
=======
what you see in Your branch
>>>>>>>
```

Modify this part in the way You deem appropriate (remember to remove the redundant symbols), and then **immediately** save and stage the changes. Move on to the next file.

7.2 With a *Merge Tool*

The *Merge Tool* can be opened either by selecting the conflicting files in Git Staging or right-clicking the project name in the *Package Explorer* and choosing:

`Team` → `Merge Tool`.

There are two main sections in the *Merge Tool*. The left one shows what the files look like in the branch You're merging into. This is where You apply Your changes. The right one shows what the files look like in Your branch. You can't change anything here.

The *Merge Tool* highlights all differences between the two files. Use it to Your advantage:

- **Blue** - the change exists in Your branch, doesn't exist in the branch You're merging into;
- **Green** - the change doesn't exist in Your branch, exists in the branch You're merging into;

- **Red** - both Your branch and the branch You're merging into have applied different changes to the same segment, and they can't be combined together;
- **Grey** - both Your branch and the branch You're merging into have applied different changes to the same segment, but they can be combined together.

Every time You finish applying changes to the given file, **immediately** save and stage them. Once You're done, commit the changes. The merge commit message will be generated automatically.

8 Git commit messages

I know it's hard, but it is very important to make commit messages as consistent as possible. Try to follow these few simple guidelines:

- Start the message with the imperative form of the verb i.e. *add, fix, delete*;
- Begin with a capital letter;
- Keep it short;
- Provide the concise outline of what You did;
- Don't end the message with a period.

9 Contributing

If You want to contribute, check out the GitHub repository.

10 Acknowledgements

I would like to thank Zoya Anwar for her invaluable feedback.