10 Git Commands for Collaborating

Learn Just Enough Git and GitHub to Contribute to an Open Source Project

Jim Tyhurst

2021-03-14

Abstract

This is a short tutorial to learn just enough Git and GitHub to contribute to an open source project with confidence. I explain how you only need to learn 10 commands and a few workflows, in order to use Git to collaborate with others.

Contents

1 Introduction

1.1 You want to contribute to an open source project

This tutorial provides a concise explanation for how you can contribute to an open source project when you do not have write-rights for the project's repository. Many projects have a small team of *committers*, which are the developers responsible for maintaining the code. However, you might fix defects or you might develop enhancements that you would like to contribute back to the project. Git and GitHub provide an efficient way for you to collaborate with the project team, even if you are not a member of that team.

If you *are* a member of the team, this tutorial still provides a very concise explanation of a few workflows that will help you to manage your local and cloud repositories easily.

This tutorial is written especially for those who have tried to use Git, but have become frustrated and disillusioned. I provide diagrams that I have found to be especially useful to understand the concepts and the environment configuration that are essential for using Git effectively.

1.2 You need to work through this tutorial if . . .

- You frequently delete a local clone of a GitHub repository and clone the original again, because your local repository is hopelessly "broken" when you try to synchronize with your colleagues' changes.
- You have lost work in the past, so now you do not trust Git to manage your revisions. As a workaround, you copy files from a repository on your local machine to another "safe" directory on your local machine as an informal method of source control.
- You email files to someone on your team who knows how to add files to a GitHub repository, because you do not know how to do it yourself.
- You have played with Git a little, but you do not know how to make a Pull Request, in order to contribute to a collaborative project. In fact, the Git semantics for `pull` and `push` seem totally backwards to you.

1.3 At the end of this tutorial, you will be able to . . .

- Use the 10 most frequently used Git commands, either on the command line or in a GUI client, along with the associated development workflow, in order to participate competently in a collaborative project.
- Make changes to a project locally and then contribute those changes back to the project's repository in the cloud.
- Explain the GitHub (or GitLab or BitBucket) process to:
  - Create a Pull Request (which is a cloud operation on the host, *not* a Git command on your local machine);
  - Review, comment on, and revise a Pull Request;
  - Merge a Pull Request into the `main` branch.

1.4 Prerequisites

Before working through the main part of this tutorial, you will:

1. Install Git on a machine that you can use for this tutorial. Working through the exercises during the tutorial, rather than just reading the material, will improve your retention significantly.
2. Have an account on GitHub, a free Git repository cloud service.

1.5 Disclaimers

1. This tutorial is a very opinionated presentation. That means it teaches only *one* process that works effectively. There *are* other ways you can use Git and GitHub, but you will learn only *one* way in this tutorial.
2. We will cover a few "failure" scenarios, such as handling merge conflicts, but the main point of this tutorial is to introduce the basics of an effective workflow. Actually, merge conflicts *are* part of the normal, expected, basic workflow, so I do not consider that to be an error path.

3. Although we frequently refer to GitHub, because it is the most popular Git cloud hosting solution and has the greatest name recognition, our discussion of Pull Requests and workflow is just as applicable to other free Git cloud hosting vendors, such as:
    - BitBucket
    - GitLab
    - Framagit
    - HelixTeamHub.

1.6 10 Git commands that *will* be covered in this tutorial

1. clone
2. pull
3. branch
4. switch
5. status
6. add
7. commit
8. push
9. merge
10. stash

1.7 10 Git commands that will *not* be covered

These commands are important to learn eventually, but they are *not* essential for normal, default workflow in an on-going collaborative project:

1. init
2. config
3. diff
4. log
5. blame
6. clean
7. reset
8. rebase
9. revert
10. fetch

After the tutorial, you can browse the official Reference Manual for the details of these commands when/if you need them. Most of these commands are covered briefly in Jeff Hale's tutorial, "10 Git commands you should know".

2 Configure your copy of the source code

Most open source projects allow you to contribute to the project without being a member of their development team. A *committer* is someone who has development privileges for their source repository. A committer can *commit* changes, i.e. make changes to their source repository. A *contributor* may *propose* code changes. In this tutorial, we will assume you are a contributor, not a committer. After working through this scenario, you will see in a later section that the process is essentially the same for committers, although it is slightly simpler.

When you want to collaborate on an open source project for which you do not have commit privileges, you need to set up your own copy of the repository. You do this in two steps:

1. Fork the repository from the *project* GitHub account to *your* GitHub account.
2. Clone your repository to your local machine.

2.1 Fork

Forking a repository makes a copy. This allows you to experiment with changes without affecting the original project.

Forking happens in the cloud. You do it from a browser. Each of the Git repository cloud hosting services have their own way to do this:

- GitHub `fork` documentation
- GitLab `fork` documentation
- BitBucket `fork` documentation

2.2 Clone

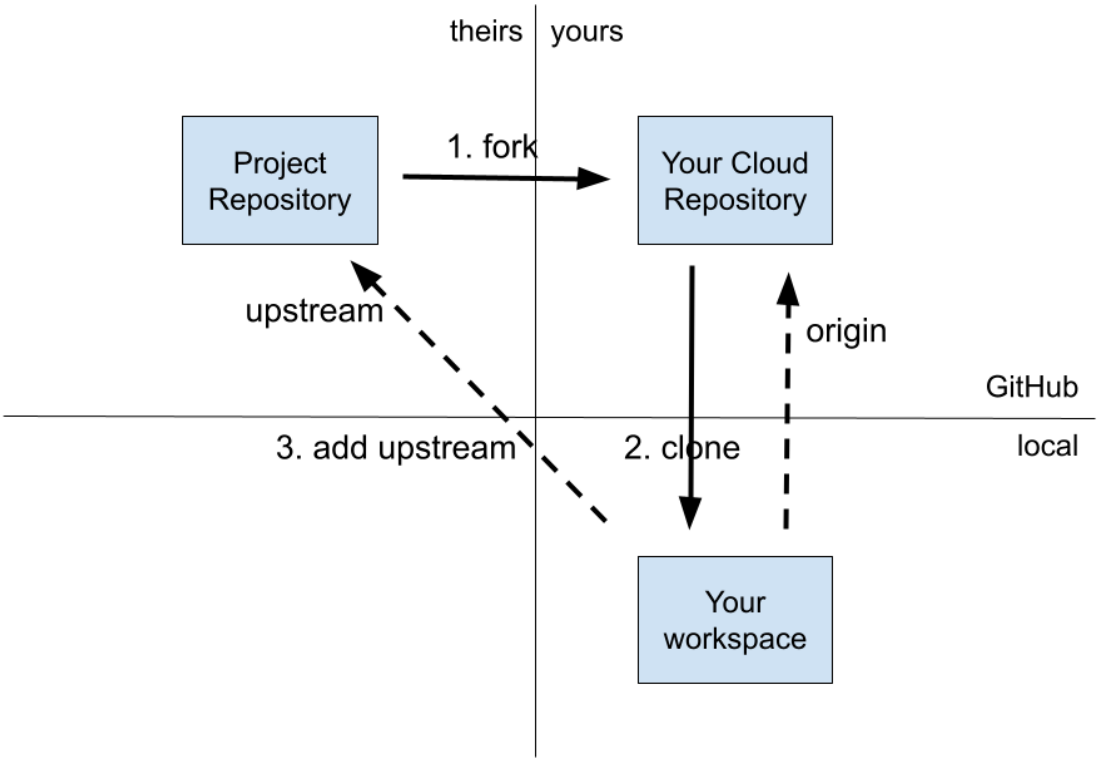
Forking creates your own copy of the repository in the cloud. In addition, you need to create a copy of the repository on your local machine, so that you can work on the code or documentation.

To get the repository to your local machine, you `clone` your cloud repository. This operation is performed on your local machine.

```
# Fork is an operation that you already performed in a browser.
# Then you clone your copy of the repository
git clone git@github.com:your-github-name/some-project.git
# Add a reference to the original source repository.
git add upstream git@github.com:project-github-name/some-project.git
```

The original project repository is referenced as `upstream`. Your fork of that repository is referenced as `origin`.

2.2.1 Fork and clone a repository



3 Keep synchronized with everyone else's changes

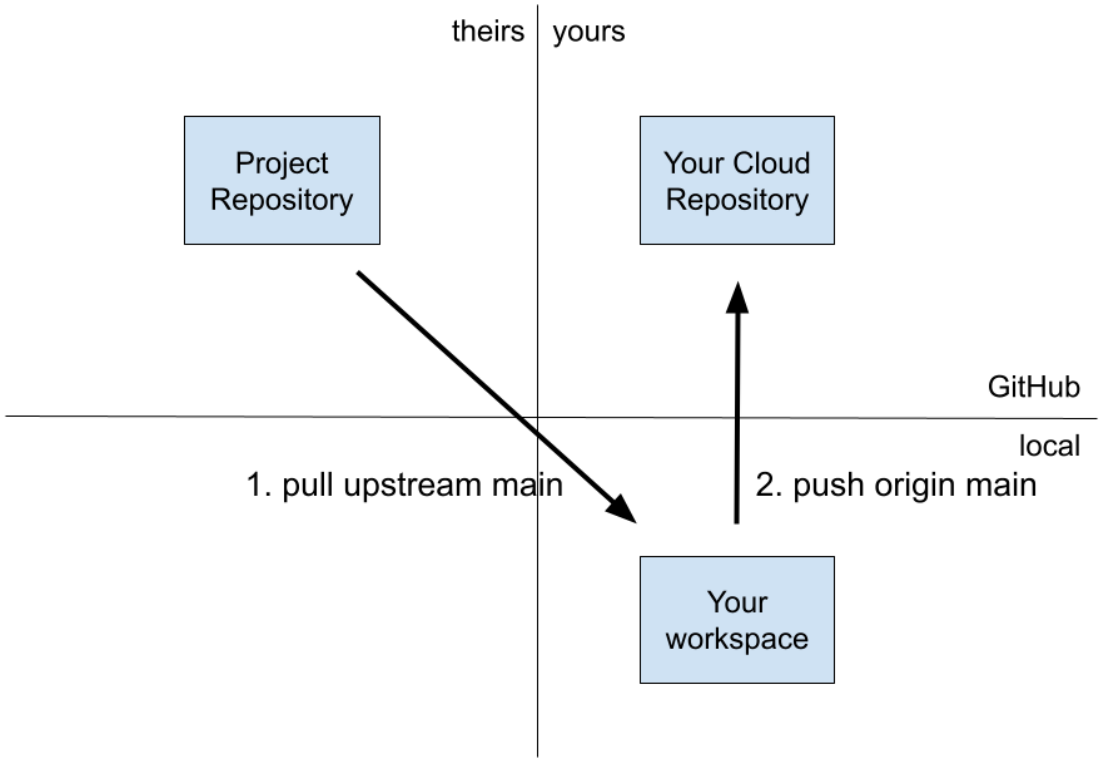
Many people will be collaborating on the project and you will want to pull other people's changes into your local repository. You do that by *pulling* changes from the `main` branch.

The pull command incorporates changes from a remote repository into the current branch.

```
git pull upstream main
```

Note that we `pull` from `upstream`, which is the project's repository, *not* from `origin`, which is your cloud copy of the project's repository.

3.1 Pull changes from the project repository



You have to synchronize your `origin` repository manually with the `upstream` repository, even though `origin` is a fork of `upstream`. GitHub does *not* keep your fork synchronized automatically for you.

4 Work in well-defined increments

In the previous sections, we set up our local repository and we saw how to keep our local repository up-to-date with other people's changes to the `upstream` project repository.

As you make changes to your local copy of the repository, you want to do that on a `branch` that will hold your proposed changes. The branch command creates a new branch of development in your local repository. By keeping your changes separate from the `main` branch, you will be able to compare your changes easily and submit your changes for review by others before the changes are merged back into the `main` branch.

The following code creates a new branch named `add-scatter-plot`, switches to that branch, and checks the status of new and used files:

```
git branch add-scatter-plot
git switch add-scatter-plot
# ... you change some files or create new files ...
# Then you check the "staged" status of files
git status
```

Different organizations choose different names for the main branch of development. In addition to `main`, other common names are `master`, `trunk`, or `prod` (short for *production*). There are also different conventions for naming branches. Be sure to check your project's documentation before you start making branches. Many open source projects have a `CONTRIBUTING.md` document in the top-level directory, explaining their project's rules and conventions.

The `status` command shows you files that differ from the committed state of your source tree, which are those files that:

- are staged and will be included when you run the `commit` command;
- have changed, but have not yet been staged for a `commit`; or
- are not yet tracked by Git. If these untracked files *should* be included in the version control system, then you need to `add` them (see the next section).

4.1 Command Reference

- branch
- switch
- status

5 Submit your proposed changes for review

After making some changes to your code base, you want to make your changes available for review, so that they can be incorporated into the project repository. First, you `commit` the changes to your local repository. Then you `push` the changes to your cloud repository (`origin`). Then you *create a pull request*, offering your changes to the project repository (`upstream`). At that point, the project team reviews your changes. They may suggest changes, in which case you make another round of changes, commit, and push those changes to `origin`. GitHub incorporates this second change set automatically into your original pull request.

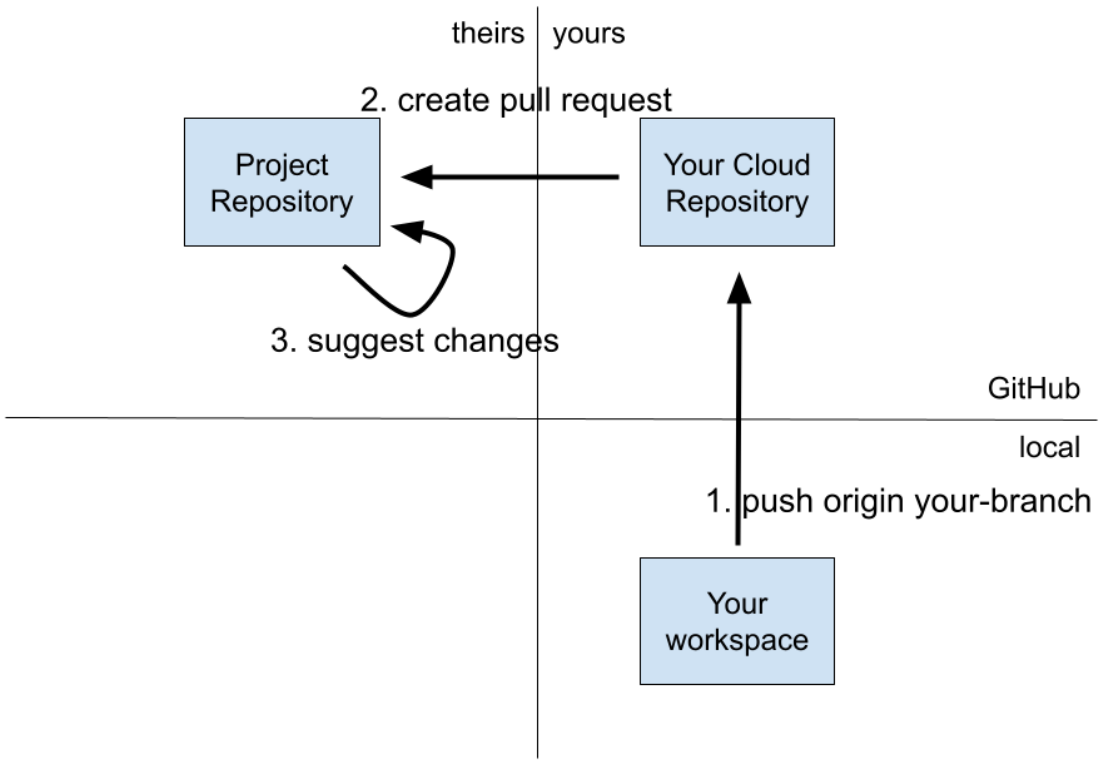
The following code adds all of the changed and new files from the current directory to the "staged" files, then commits those staged files, and pushes the `add-scatter-plot` branch to the `origin` repository:

```
git add .
git commit
git push origin add-scatter-plot
# You create a pull request from a browser
```

The example above shows a very simple case, where there is just one commit. However, usually you will go through several iterations of changing files, testing, and committing them before you push the branch for review.

Note that you `push` to `origin`, which is your copy of the repository, rather than `upstream`, which is the project's version of the repository. That is because we are assuming you are *not* a committer on the project, so you *can* change your own repository, but you *cannot* change the project organization's repository.

5.1 Push changes to your repository in the cloud



You repeat the `add`, `commit`, and `push` steps as many times as needed to respond to the reviewers'

suggestions. However, you only need to create the pull request once. After that, GitHub automatically updates the pull request each time you push that same branch to the `origin` repository.

5.2 Command Reference

- add
- commit
- push

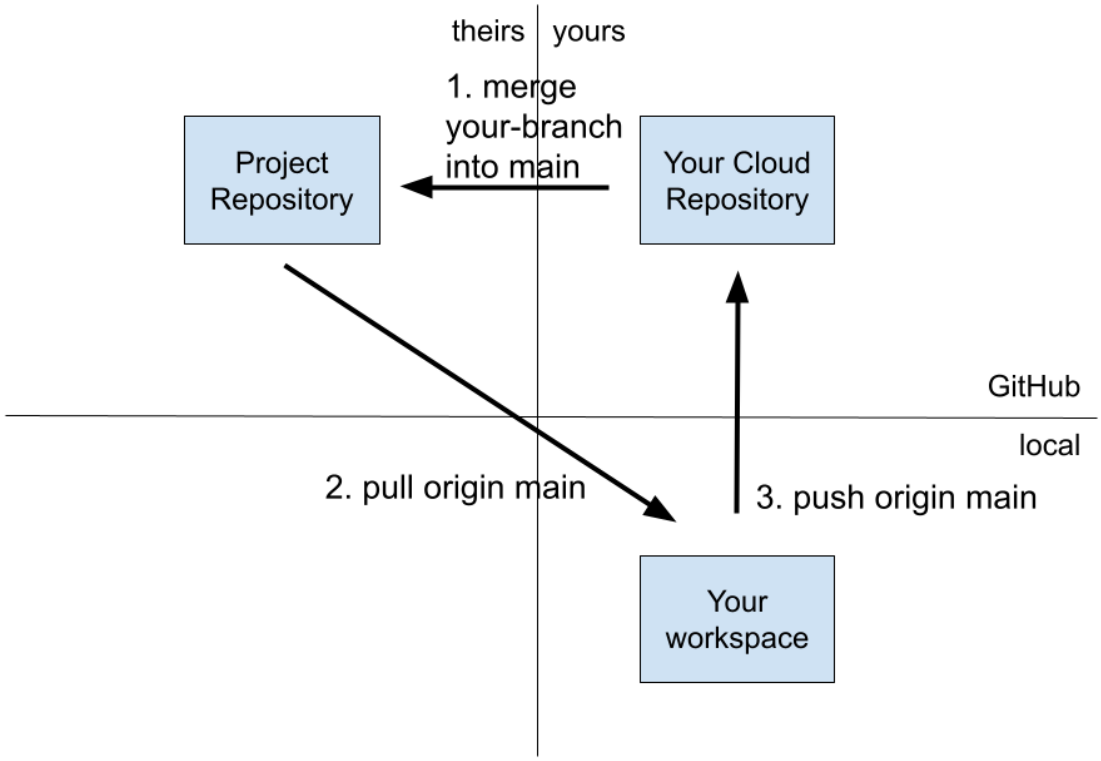
6 The project accepts your proposed change

After reviewing your proposed changes contained in a pull request, the project team may choose to merge your changes contained in a branch of the `origin` repository into the `main` branch of their project repository. This merge is accomplished through the GitHub UI in the browser. Once your code as been incorporated to the project repository, you need to synchronize with that repository.

The following code pulls the most recent code from the `upstream main` branch. Remember to push those changes to your `origin` as well, so that your cloud repository stays synchronized with the project repository. You have already seen these three commands {`select`, `pull`, `push`} in previous scenarios:

```
git switch main
git pull upstream main
git push origin main
```

6.1 Pull project repository changes to your local repository



At this point, you are re-synchronized with the project's repository and you are ready to start making more changes.

It is not uncommon to need to work on two sets of changes at the same time. You can do that by working on two separate branches. All of the steps that we have seen so far work fine with you committing changes on two separate branches.

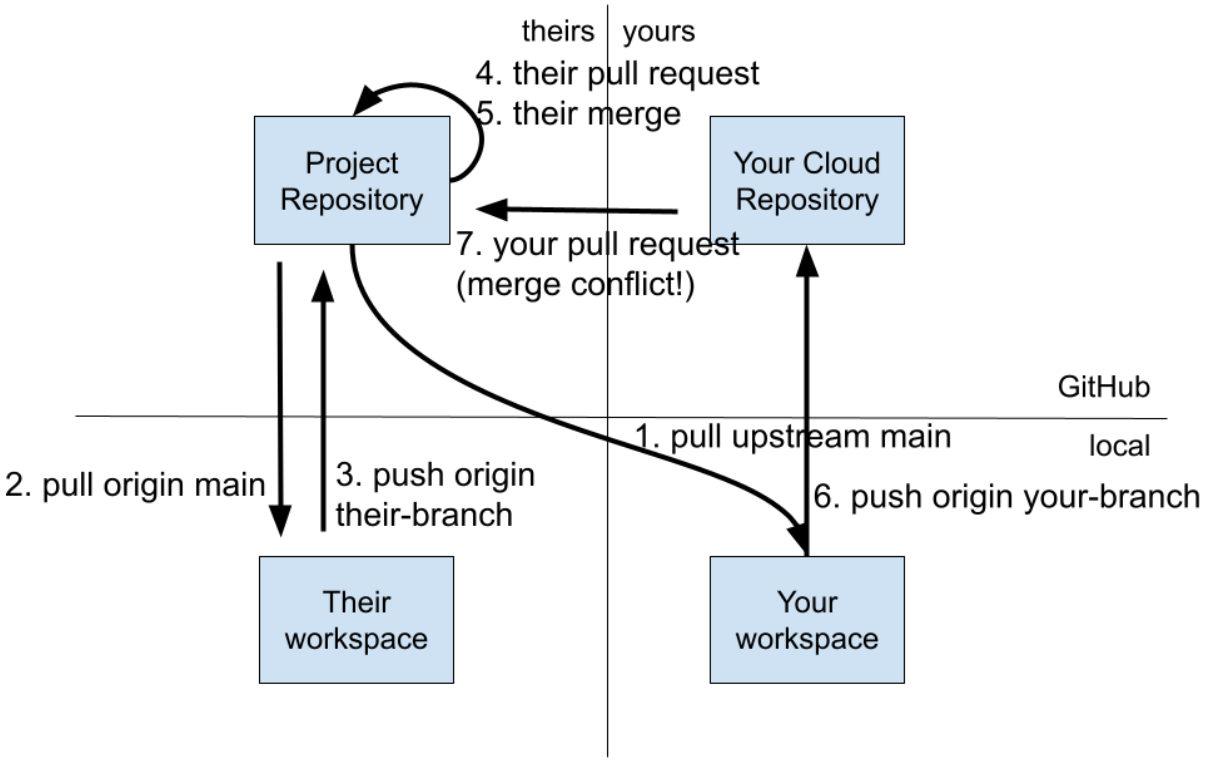
7 Oh, no! A Merge Conflict!

If you have done any work collaborating with others using GitHub, you have probably encountered a "merge conflict" at one time or another. People who are new to Git are oftened mystified when this happens, because for the most part Git does a fantastic job of merging sets of changes magically with no problems.

When two or more people are working on a code base, eventually two of you will want to make changes to the same section of code in the same file. Suppose your colleague gets their pull request merged first, so their change is merged into the `main` branch. When you come along with your pull request, Git notices that the code has changed since you last pulled `main` and it notices that you are proposing changes to the same lines of code. Therefore, it generates a merge conflict and throws it back to you to integrate your changes with the latest changes that occurred after you last pulled `main`.

The diagram below illustrates that you pull `main` as step (1). Meanwhile, as you are working on your changes, your colleague goes through a full pull, change, add, commit, push, create pull request, merge pull request cycle (steps 2-5 in the diagram). By the time you finally push `your-branch` (6) and create a pull request (7), the `main` branch in the `Project Repository` is no longer the same as the `main` branch in `Your workspace`, so there is a potential for a merge conflict.

7.1 How merge conflicts occur



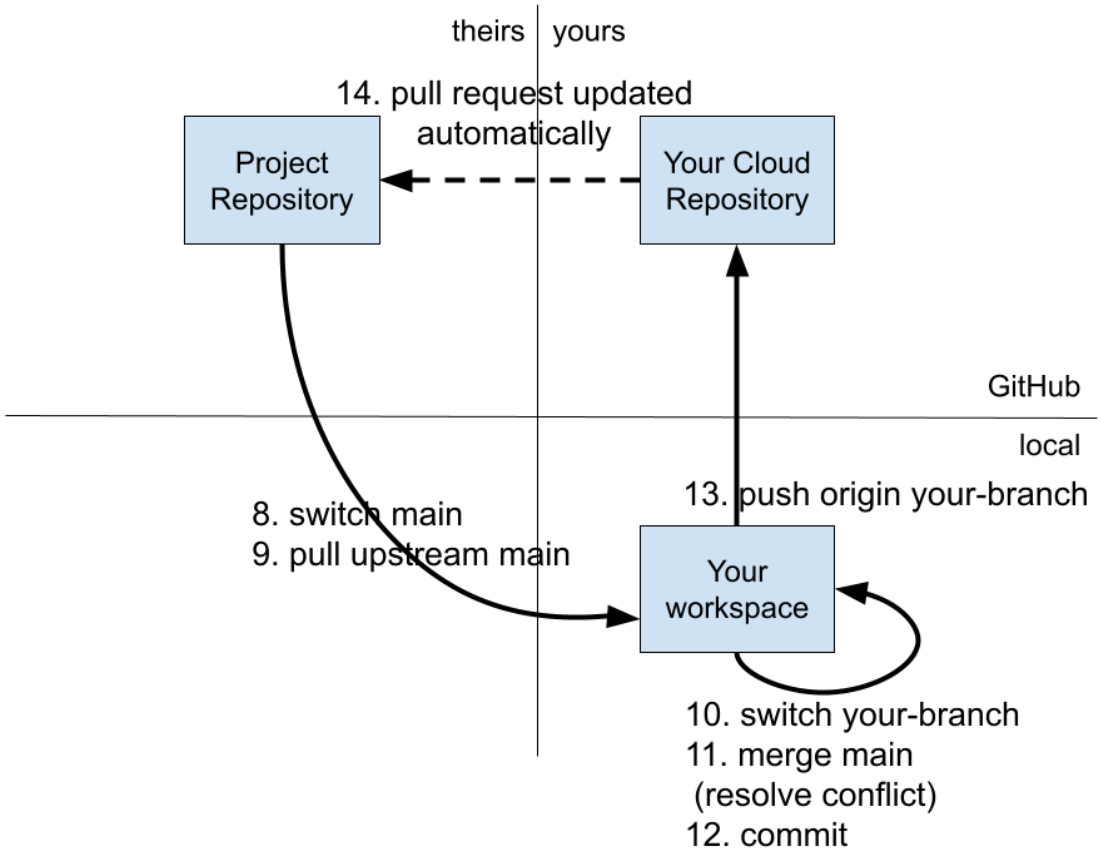
8 Resolve a Merge Conflict

When GitHub notifies you of a merge conflict, it is a warning that if you were to merge `your-branch` into `main`, the merge would fail with a merge conflict error. Therefore it is up to you to resolve the merge conflict manually, in your workspace.

```
git switch main
git pull upstream main  # There is *not* a merge conflict here!
git push origin main  # Keep your cloud repository synchronized
git switch your-branch
git merge main  # Merge main into your-branch. Merge conflict!
# (Change any files that have conflicts.)
git add .
git commit
git push origin your-branch
# GitHub automatically updates your original pull request.
```

At the end of this sequence, your pull request probably does not have a merge conflict, *unless* another colleague got their pull request approved for the same block of code while you were trying to resolve the first merge conflict. In which case, you go through the process again to fix this next merge conflict.

8.1 Resolve a Merge Conflict



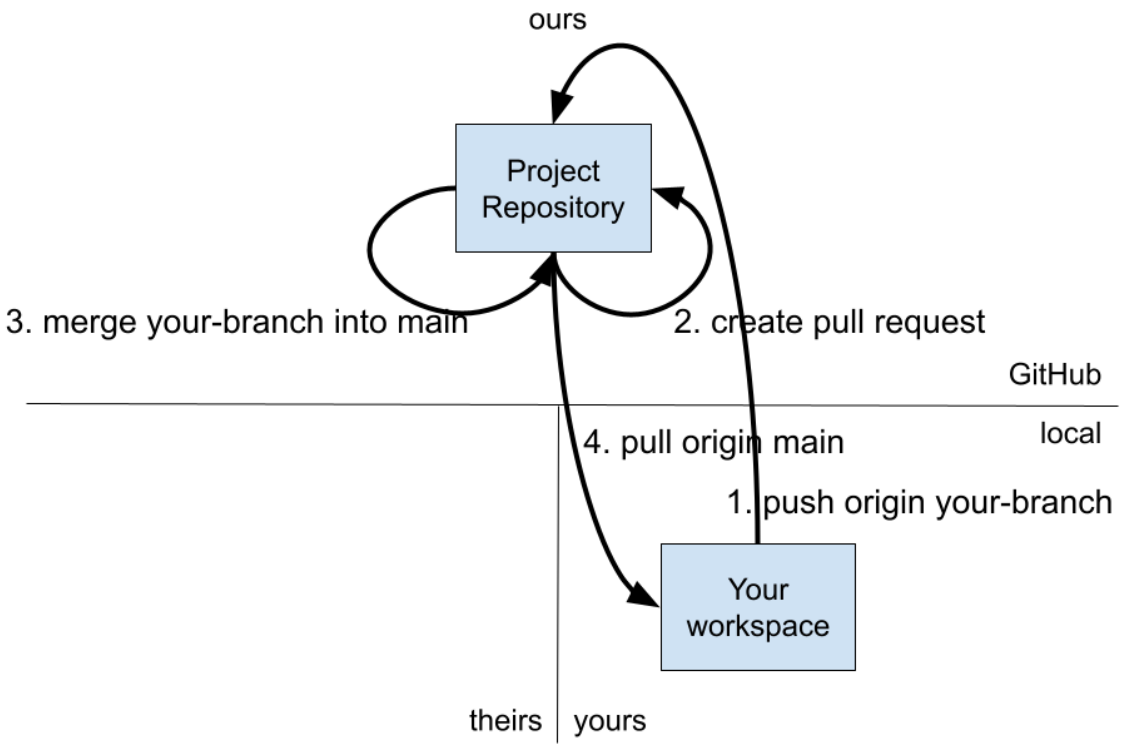
9 Working as a Committer

When you have permissions to work in the `Project Repository`, then your local workspace only references `origin` without a reference for `upstream`. The rest of the processes that we have considered are fundamentally the same.

The normal flow is:

```
git switch main
git pull origin main
git branch your-branch
git switch your-branch
# (Make some changes.)
git add .
git commit
git push origin your-branch
# (Create pull request on GitHub.)
# (Accept pull request on GitHub.)
git switch main
git pull origin main
git branch your-next-branch
# ...
```

9.1 Workflow without a forked repository



10 I forgot to make a branch!

You know that you are supposed to do new work on a branch, but sometimes you forget to create a new branch, so you start working accidentally on `main`. You can use the stash command to fix this situation, as shown below.

```
git switch main
# (Make some changes.)
# Oops! I wanted these changes on a branch!!
git stash push
# Your changes are no longer on main, but they have been saved.
git status  # Verify that your changes are gone.
git branch your-branch
git switch your-branch
git stash pop
# Your previous changes are now on your-branch.
# (Make more changes.)
# (Continue as usual ...)
```

11 Annotated References

11.1 Reference material

- Git Reference Manual.
  https://git-scm.com/docs
  Look here first when you need to learn a command. Then you can do a browser search if you need more discussion of problems that you encounter.
- Scott Chacon and Ben Straub. 2014. Pro Git. 2nd Edition.
  https://git-scm.com/book/en/v2
  This published paper book is also available for free online. Useful for reference or as an extended tutorial.
- Tobias Günther. July 2020. 17 Ways to Undo Mistakes with Git.
  https://www.git-tower.com/blog/surviving-with-git-videos
  Excellent, easy to follow instructions for making things right when you mistakenly applied a command in the wrong place.

11.2 Tutorials

- The official Git tutorial.
  https://git-scm.com/docs/gittutorial
  A very brief introduction to common Git commands.
- GitHub Workflow.
  https://www.youtube.com/watch?v=47E-jcuQz5c
  A very short 1 min 13 sec video from GitHub, which succinctly summarizes the `branch > commit > Pull Request > merge` workflow.
- Jenny Bryan, et al. "Happy Git and GitHub for the useR".
  https://happygitwithr.com/
  Free, online R-focused gentle tutorial that starts from the very basics for using Git for your R projects. It is also useful as a reference when you get stuck in a particular situation. This tutorial provides very thorough coverage of situations that you will encounter while using Git to develop software.
- Learn Git with Bitbucket Cloud.
  https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud
  This tutorial is especially good if you are using BitBucket, although most of it generalizes to other hosting platforms also.
- Git Feature Branch Workflow.
  https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow
  Excellent tutorial from Atlassian explaining how to use feature branches in your development process. The tutorial is useful whether or not you use Atlassian's Bitbucket as your free, cloud-based repository.
- Jeff Hale. 2019-02-28. 10 Git Commands You Should Know: Plus tips to save time with Git.
  https://towardsdatascience.com/10-git-commands-you-should-know-df54bea1595c
  Provides an introduction to some of the Git commands that were not covered in the previous sections of my tutorial.
- Scott Chacon and Ben Straub. 2014. Pro Git. 2nd Edition.
  https://git-scm.com/book/en/v2

This published paper book is also available for free online. Useful for reference or as an extended tutorial.

11.3 Graphical User Interfaces

- Sourcetree.
  https://www.sourcetreeapp.com/
  Highly recommended as a free, visual Git client for Mac or Windows. Sourcetree is *very* helpful for visualizing the branch structure of your repository. I prefer to run Git commands on the command line, but I still use Sourcetree to compare commits and to visualize the relationships between branches.
- List of GUI Clients.
  https://git-scm.com/downloads/guis
  A useful list if you prefer to work with a graphical user interface (GUI), rather than the command line.

11.4 Good practices

- Conventional Commits: A specification for adding human and machine readable meaning to commit messages.
  https://www.conventionalcommits.org/
  This specification provides a standard for your commit messages, so that release documentation can be generated automatically.

11.5 Free cloud-based repository hosts

- GitHub
- BitBucket
- GitLab
- Framagit, a deployment of GitLab, managed by FramaSoft, a non-profit educational organization whose motto "Degooglize your Internet" is backed up by many useful free, open, cloud-based collaborative services, offered as alternatives to Google's many applications.
- HelixTeamHub