'3rd Online Training "make your data FAIR": Version Control with Git and GitHub'

Stefano Rapisarda (RDM, Utrecht University Library)

2024-10-15

Table of contents

1 Setup Instructions

Disclaimer: during the workshop, you will go through the basic steps of creating a Git repository, committing changes in your repository, and uploading your repository to GitHub. If you want to follow along with the instructor, you need to follow the following setup steps. You are free to just be a "spectator" and then set up Git and GitHub at your own pace, as the instructor will provide you with the basic instructions to perform those first basic steps.

1.1 Set up Git

Warning: If your operating system is either macOS or Linux, you most likely have Git already installed. Open a terminal and type `git --version`. If you already have Git installed, your terminal will print a line specifying your Git version. If you don't have Git installed, follow these steps:

1. **Check the Git installation page** and follow the instructions according to your operating system: Installing Git.
2. **For Windows users**: Install Windows Terminal from the Microsoft Store.
3. Open a terminal (**For Windows users**: Use Windows Terminal, NOT the command prompt) and type `git --version`. If Git is installed, you should see a line indicating your current Git version.
4. **Set up a GitHub account** by following part 1 of this guide: Getting Started with Your GitHub Account.

1.2 GitHub First-Time Push and Authentication Guide

As explained during the workshop, one of the most powerful features to exploit when using Git and GitHub is connecting your local repository to a (GitHub) remote repository.

In order to do that, you need a local repository (a project created in your computer) and an empty remote repository created on GitHub or any other hosting service. From now own, we will assume you are using GitHub.

When you create a repository on GitHub, you can define the level of security of your repository in the Settings of your GitHub account. If you don't have any specific security preference, you will leave GitHub to use default settings. In most cases, the workflow define below should be enough to connect your local and remote repository, but keep in mind that some extra steps could be needed if the remote repository has been created by someone else or it is related to an institution with its own identification/login procedure (via some sort of student/employer ID, for example)

1. Link Local Repository to GitHub

- On the GitHub repository page, copy the URL of your repository.
- In your terminal, add the remote origin:

```
git remote add origin <repository-url>
```

2. Set Up Authentication

GitHub now prefers personal access tokens over passwords for authentication. Here's how to set it up:

- Generate a Personal Access Token (PAT):
  - Go to GitHub Settings > Developer settings > Personal access tokens.
  - Click "Generate new token".
  - Give it a name, set an expiration, and select necessary scopes (at least 'repo').
  - Click "Generate token" and copy the token immediately.
- Configure Git to use the token:
- Option 1: Use Git Credential Manager (recommended for Windows and macOS):
  - It should prompt you for authentication on your first push.
  - Enter your GitHub username and use the PAT as the password.
- Option 2: Store the PAT in your system's credential store: `git config --global credential.helper store` On your first push, you'll be prompted for your username and password (use PAT).
- Option 3: Use the token in the remote URL (less secure): `git remote set-url origin https://<username>:<token>@github.com/<username>/<repository>.git`

3. Push Your Code

Now you're ready to push your code:

```
git push -u origin main
```

(Use 'main' or 'master' depending on your default branch name)

4. Two-Factor Authentication (2FA)

If you have 2FA enabled on your GitHub account:

- When prompted for authentication, use your GitHub username and PAT.
- You won't need to enter a 2FA code when using a PAT.

5. Subsequent Pushes

For future pushes, if you've set up authentication correctly, you should be able to push without re-entering credentials:

```
git push
```

1.2.1 Troubleshooting

- If you encounter authentication issues, ensure your PAT hasn't expired and has the correct permissions.
- For SSH users, make sure you've added your SSH key to your GitHub account.

Remember to keep your PAT secure and never share it publicly. Treat it like a password.

1.3 Before the Workshop

- Keep a terminal open so that you are ready to type commands;
- Keep your browser open on your GitHub account page.

2 Git commands used during the workshop

git init

- **Use Case:** Starting a new project
- **Description:** Creates a new Git repository in the current directory
- **Common Options:** None
- **Example:** `git init`

git clone

- **Use Case:** Getting a copy of an existing project
- **Description:** Creates a local copy of a remote repository
- **Common Options:**
  - `--branch <branch-name>`: Clone a specific branch
  - `--depth <depth>`: Create a shallow clone with limited history
- **Example:** `git clone https://github.com/user/repo.git`

git add

- **Use Case:** Preparing changes for a commit
- **Description:** Adds file changes to the staging area
- **Common Options:**
  - `.`: Add all changes in the current directory
  - `-p`: Interactively choose parts of files to add
- **Example:** `git add file.txt` or `git add .`

git commit

- **Use Case:** Recording changes to the repository
- **Description:** Creates a new commit with the staged changes
- **Common Options:**
  - `-m "message"`: Specify the commit message
  - `-a`: Automatically stage all modified files
- **Example:** `git commit -m "Add new feature"`

git push

- **Use Case:** Sharing local commits with a remote repository
- **Description:** Sends local commits to a remote repository
- **Common Options:**
  - `-u origin <branch>`: Set up tracking for a new branch
  - `--force`: Force push (use with caution)
- **Example:** `git push origin main`

git pull

- **Use Case:** Updating local repository with remote changes
- **Description:** Fetches changes from a remote repository and merges them
- **Common Options:**
  - `--rebase`: Rebase local changes on top of remote changes
- **Example:** `git pull origin main`

git branch

- **Use Case:** Managing branches
- **Description:** Lists, creates, or deletes branches
- **Common Options:**
  - `-d <branch-name>`: Delete a branch
  - `-a`: List all branches (local and remote)
- **Example:** `git branch feature-branch`

git checkout

- **Use Case:** Changing branches or restoring files
- **Description:** Switches to a different branch or restores files
- **Common Options:**
  - `-b <branch-name>`: Create and switch to a new branch
  - `-- <file>`: Restore a file to its last committed state

- **Example:** `git checkout feature-branch`

git merge

- **Use Case:** Integrating changes from different branches
- **Description:** Merges changes from one branch into another
- **Common Options:**
  - `--no-ff`: Create a merge commit even if fast-forward is possible
- **Example:** `git merge feature-branch`

git status

- **Use Case:** Checking the state of the working directory
- **Description:** Shows the status of changes in the working directory
- **Common Options:**
  - `-s`: Give output in short format
- **Example:** `git status`

git log

- **Use Case:** Viewing commit history
- **Description:** Shows the commit history of the repository
- **Common Options:**
  - `--oneline`: Show each commit on a single line
  - `--graph`: Display an ASCII graph of branch and merge history
- **Example:** `git log --oneline --graph`

3 Git and GitHub Glossary

Repository (Repo) A container for a project that contains all the files, folders, and version history. It can be local (on your computer) or remote (on a server like GitHub).

Staging Area (Index) A intermediate area where changes are prepared before committing. It allows you to group related changes into a single commit.

Commit A snapshot of your repository at a specific point in time. Each commit has a unique identifier and includes a message describing the changes.

Branch A parallel version of the repository that allows you to work on different features or experiments without affecting the main codebase.

Pull Request (PR) A GitHub feature that allows you to propose changes from one branch to another. It's often used to review code before merging it into the main branch.

Version Control A system that records changes to files over time, allowing you to recall specific versions later. Git is a distributed version control system.

Clone Creating a local copy of a remote repository on your machine.

Fork A personal copy of someone else's repository on GitHub. It allows you to freely experiment with changes without affecting the original project.

Merge The process of combining changes from one branch into another.

Remote A version of your repository that is hosted on a server (like GitHub). You can have multiple remotes.

Push Uploading your local repository changes to a remote repository.

Pull Downloading changes from a remote repository to your local repository and merging them with your local files.

Fetch Downloading changes from a remote repository to your local repository without merging them.

Head A reference to the most recent commit in the current branch.

Master/Main The default primary branch in a repository. Traditionally called "master", many projects now use "main".

.gitignore A file that specifies which files or directories Git should ignore and not track.

Conflict Occurs when Git can't automatically merge changes from different branches. Requires manual resolution.

Tag A reference to a specific point in the repository's history, often used to mark release points.

Rebase Reapplying commits on top of another base commit. It's an alternative to merging for integrating changes from one branch into another.

Origin The default name Git gives to the server you cloned from.

Upstream Typically refers to the original repository that you forked from, especially in the context of open-source projects.

4 Selected Resources

1. **Pro Git Book**
   - Description: Comprehensive guide to Git by Scott Chacon and Ben Straub
   - Purpose: In-depth learning of Git from basics to advanced topics
   - Link: https://git-scm.com/book/en/v2
2. **GitHub Guides**
   - Description: Official GitHub documentation and tutorials
   - Purpose: Learn GitHub-specific features and workflows
   - Link: https://guides.github.com/
3. **Atlassian Git Tutorials**
   - Description: Beginner-friendly Git tutorials by Atlassian
   - Purpose: Step-by-step learning of Git concepts and commands
   - Link: https://www.atlassian.com/git/tutorials
4. **Git - The Simple Guide**
   - Description: Simple, straightforward guide to Git basics
   - Purpose: Quick reference for common Git commands and workflows
   - Link: https://rogerdudler.github.io/git-guide/
5. **Learn Git Branching**
   - Description: Interactive visualization tool for learning Git branching
   - Purpose: Practice Git branching strategies through gamification
   - Link: https://learngitbranching.js.org/
6. **GitHub Skills**
   - Description: Interactive courses for learning GitHub
   - Purpose: Hands-on practice with GitHub features and workflows
   - Link: https://skills.github.com/
7. **Git for Computer Scientists**
   - Description: Quick introduction to Git internals
   - Purpose: Understand Git's underlying concepts and data structures
   - Link: https://eagain.net/articles/git-for-computer-scientists/
8. **Git from the Bottom Up**
   - Description: Detailed explanation of Git's core concepts
   - Purpose: Deep dive into how Git works under the hood
   - Link: https://jwiegley.github.io/git-from-the-bottom-up/

9. **Oh Shit, Git!?!**
    - Description: Guide to recovering from common Git mistakes
    - Purpose: Troubleshooting and fixing Git errors
    - Link: https://ohshitgit.com/
10. **Git Cheat Sheet**
    - Description: Visual cheat sheet for Git commands
    - Purpose: Quick reference for Git commands and workflows
    - Link: https://education.github.com/git-cheat-sheet-education.pdf
11. **GitHub YouTube Channel**
    - Description: Official GitHub video tutorials and webinars
    - Purpose: Visual learning of GitHub features and best practices
    - Link: https://www.youtube.com/github
12. **GitKraken Git Client**
    - Description: Powerful Git GUI client with built-in tutorials
    - Purpose: Visual Git management and learning
    - Link: https://www.gitkraken.com/
13. **GitHub Blog**
    - Description: Official GitHub news and feature announcements
    - Purpose: Stay updated on GitHub features and best practices
    - Link: https://github.blog/
14. **Git Best Practices**
    - Description: Compilation of Git best practices by Seth Robertson
    - Purpose: Learn advanced Git usage and team workflows
    - Link: https://sethrobertson.github.io/GitBestPractices/
15. **GitHub Docs**
    - Description: Comprehensive documentation for all GitHub features
    - Purpose: In-depth reference for GitHub functionality
    - Link: https://docs.github.com/en

5 Questions during the workshop

How to use two-factor-authentication?

Since March 2023, GitHub made two factor authentication mandatory for accessing GitHub repositories (in the past, you could simply access with your username and password, one factor authentication, 1FA).

You can choose among a range of options for your 2 factor authentication and in some cases (for example when your repository has been created by your institution) this could already been implemented.

You can find everything you need to know about setting up two-factor-authentication here.

You described now the technical part of version control and snapshots. Can you share insights on the systematic approach in terms of where it makes sense to take a shot? (if this question is not to specific)

In short, every time you think your changes represent a **significant** change in your project, then it is time to commit, i.e. to take a snapshot. You can also think in terms of snapshot description, think about how you would describe the commit first and try to determine if it is a change "deserving" an indipendent commit. For example:

- Correcting a typo in the documentation –> no indipendent commit is needed, this change can be part of a series of changes related to a documentation review. You can commit when, for example, you reviewed an entire paragraph. Your commit message would then be "reviewd 1st paragraph, typos and grammar corrected";
- Adding a new function –> This is something that would definetively need its own commit. If you submit this change for review as a pull request (together with other commits), it would be also nice for the person in charge of reviewing the code to visualise a list of nicely organised commits with clear description. So think also about other people visualising your development history, you don't want them to got through thousands of very small, not very meaningful, commits;
- Finding and correcting a bug –> This would also need its own commit for the reasons explained above.

I just have a question regarding the branches. When we close a branch, is it deleted from the version control overview?

When working with Git and GitHub, it's important to understand what happens to branches after they're merged. Contrary to what some might expect, merging a branch doesn't automatically remove it from your repository's history or the list of branches. Remember, when dealing with Git and GitHub, do not assume that actions are taken automatically, you have to be explicit about all the steps!

In Git, when you merge a branch, say 'feature-x', into your main branch, the 'feature-x' branch still exists. It's simply that the main branch now includes all the changes from 'feature-x'. You can continue to see 'feature-x' in your list of branches if you run 'git branch'. For example:

```
$ git checkout main
$ git merge feature-x
$ git branch
  feature-x
* main
```

As you can see, 'feature-x' is still there. If you want to remove it locally, you need to do so explicitly:

```
$ git branch -d feature-x
```

GitHub works similarly, but it adds a layer of project management on top. When you merge a pull request in GitHub, the platform marks the pull request as "closed", but this doesn't delete the associated branch. The branch remains in your repository's history and can still be viewed or restored.

GitHub does offer an option to automatically delete head branches after merging a pull request. In our example during class, you could see a button with the option of deleting the branch after merging. This is a repository setting, not a Git feature. If enabled, it can help keep your branch list clean, but it's important to note that this deletion is reversible. GitHub allows you to restore recently deleted branches.

Even when branches are deleted, either manually or automatically, the commit history isn't erased. Git's data model ensures that all commits remain in the repository's history. This means you can always recover a deleted branch if needed. For instance, if you accidentally delete a branch, you can often recover it using Git's reflog:

```
$ git reflog
$ git checkout -b recovered-branch <SHA>
```

In practice, teams often develop different conventions for managing branches post-merge. Some prefer to keep branches for reference or potential future work, while others delete them promptly to maintain a cleaner workspace. For example, a team might decide to keep feature branches for a week after merging before deleting them, allowing time for any issues to surface while still eventually cleaning up.

It's also worth noting that local and remote branches are managed separately. Deleting a branch locally doesn't affect the remote repository, and vice versa. To delete a remote branch, you'd use a command like:

```
$ git push origin --delete feature-x
```

In essence, Git and GitHub provide flexibility in how you manage your branches after merging. They don't automatically remove branches, allowing you to decide when and if to delete them based on your team's workflow and preferences. This approach preserves history and allows for easy reference or restoration of past work, while also providing tools to keep your repository organized when needed.