Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Report

Git

written by

Jan Moritz Witt

Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik
Arbeitsbereich Wissenschaftliches Rechnen

Studiengang: Meteorologie
Matrikelnummer: 7174687

Betreuer: Dr. Hermann Lenhart

Hamburg, 2020-07-18

Contents

1 What is Git? 4
1.1 What is Version Control? 4
1.2 What makes Git so popular? 6
1.3 Working with branches in Git 7

2 Collaboration on one project 10
2.1 Remote Repository 10
2.2 GitHub 11

One remark for the start: Because Git is a very powerful tool for version control with a lot of unique commands it is not possible to cover everything in one presentation and report. For everyone with further questions who starts with Git I recommend [1]. It was the source for most of the information presented here and is very beginner-friendly.

1 What is Git?

When working on a project the worst thing that can happen is that on some way the work gets lost. This can happen very spontaneously because of a lot of reasons like the memory of the computer breaks or by overwriting something accidentally. Or the user works on a problem and after several hours has to realize, something seemed unnecessary or not as the right way to do it but now would be the perfect solution but got removed already. Without Git or another Version Control System (VCS) all the work has to be done all over again.

1.1 What is Version Control?

Version Control Systems offer the opportunity to record changes to a file or set of files so you can recall any specific version of it later. This can be done with any type of file, e.g. images, layouts, videos or text files. It is also possible for entire projects with several contributors. For every change VCS save the time and the creator of the change, therefore in case of problems in the integrations in the projects the source can be found very easily. VCS can be divided into three groups; Local VCS, Centralized VCS (CVCS) and Distributed VCS (DVCS).
The local Version Control Systems were the first that occurred and as the name indicates describes a VCS on the same local machine as the real projects. It started with a simple copy into another directory, the clever ones with a time-stamped directory. The disadvantage of another directory is that it is easy to mix up the projects and its copy. An example for a local VCS is Revision Control System (RCS). It keeps sets of patches in a special format on a disk. A special version can be recreated by adding up all previous versions.
After local VCS the centralized VCS were invented. The project is hosted on a single server which contains all versions. Now several contributors were able to have access to the data which enabled collaboration in the first place. Furthermore the contributors are able to see what the other are doing so the risk of overlapping work gets reduces. Every work on the projects can be easily supervised by an administrator, who has full access control to the server. Prominent examples of centralised VCS are Concurrent Version Control System (CVS), its successor Subversion (SVN) and Team Foundation Version Control (TFVS) from Microsoft. On the down side, if the server is down, no one can work on the project and if the server get somehow corrupted everything is lost when there is no proper back-up. [2]
Which led to the development of the third group of VCS, the distributed ones (DVCS). Every contributor mirrors the entire repository of the project, i.e. all files and all previous

versions of them, on its local machine. These copies are comprehensive enough that each can be used to recover the original if it gets corrupted. Every contributor can work simultaneously and independently and synchronises the local repository with the original repository to share the work with the co-workers and download their work. Examples are Mercurial, Bazaar and Git.

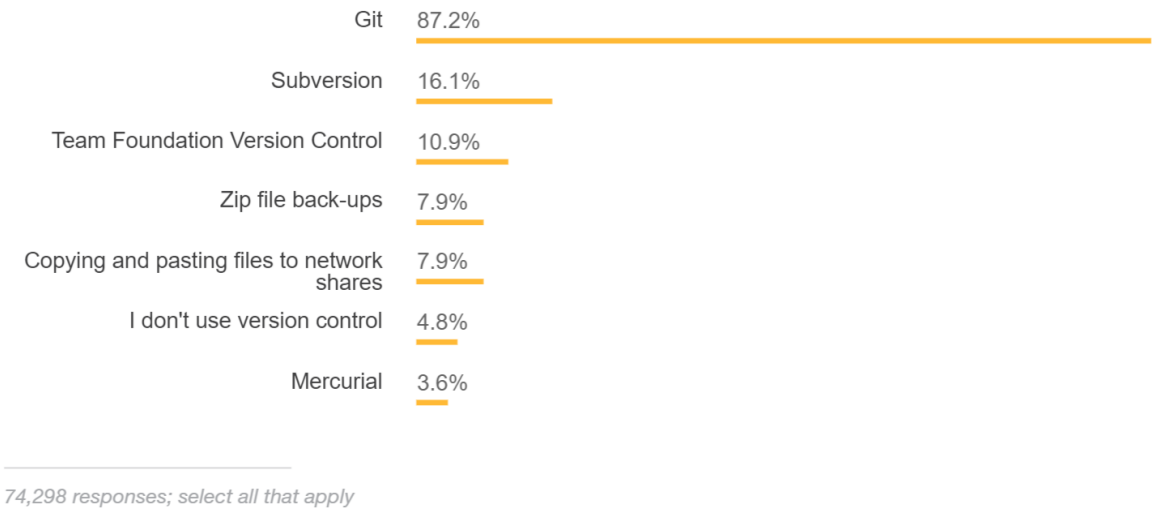

Figure 1.1: Result from a Survey made by Stackoverflow in 2018 [3]. The question was which version control is used by the users. When adding the percentages it exceeds 100% which leads to the conclusion that several users use more than one version control.

While their are several options of version control systems figure 1.1 shows very clear that in practise nearly 90% of all participants on the StackOverflow survey in 2018 used Git as version control system [3].

Using Version Control

When a new repository of a project is created, it is empty. So far nothing is saved and all files of the projects are untracked, i.e. no back-up with for example Git exists. To track a file the user has to deliberately add it to the Git repository. The reason why not all files of a folder or project are directly tracked by Git is that their might be files that shouldn't be shared with collaborator, e.g. which contain information about the local computer, or files that contain no helpful information for the project and might just irritate other users.
After a file is tracked, Git recognizes automatically if the version in the working area is exactly the same as the back-up version in the repository or if it has been modified. If the file has been modified the current version is not saved in the repository yet. When the modification is complete and the user wants to update the repository, the file gets staged. This means with the next commit this version of the file gets saved into the repository.

The staging area makes it possible to combine several modifications targeting the same issue in one commit. A commit is the next snapshot of the state of the project in the history of the repository. With a commit all staged files gets saved into the repository and the user can later load this version again from the repository. The changes are saved with useful comments to make the reasons behind them comprehensible even after a long time after the commit or for collaborators which are not familiar with the issue.
The cycle of unmodified, modified, staged and again unmodified after the commit are the states of a tracked file in the repository. It is done over and over again to create the file history in the repository and is most basic way to use a VCS.

1.2 What makes Git so popular?

To understand this survey result, a deeper look into Git, its history and a comparison to other VCS is necessary.
Git was not one of the first distributed version control systems. In 2005 the inventor of Linux, Linus Torvalds, decided to create its own tool-based DVCS after the relationship to the former DVCS of Linux, Bitkeeper, broke down. Due to his experience he had several goals in mind how the implementation should look like:

- Speed
- Simple design
- Strong support for non-linear development (parallel branches)
- Fully distributed
- Able to handle large projects like the Linux kernel efficiently

With this goals in mind, Git has several different features to other DVCS. While most DVCS save the changes as a list file-based changes, which is often called delta-based version control, Git stores the history of the repository as a series of snapshots of the entire system. In addition, most operations can be done directly on the local files so no connection to the original repository is necessary which leads to no network latency overhead.

Comparing the time needed for several operations for Git and Subversion in figure 1.2 shows the advantage very clear. Git is faster for every of the shown operations.

For instance, a basic update of a file in the repository (Commit Files(A)) is faster by the factor of 4 than Subversion. However, it has to be kept in mind that this also the comparison between centralised and distributed version control systems. Therefore it is not very surprising that taking a look into all changes to the repository (Log(All)) Git is much faster (factor 325).

Operation		Git	SVN	
Commit Files (A)	Add, commit and push 113 modified files (2164+, 2259-)	0.64	2.60	4x
Commit Images (B)	Add, commit and push a thousand 1 kB images	1.53	24.70	16x
Diff Current	Diff 187 changed files (1664+, 4859-) against last commit	0.25	1.09	4x
Diff Recent	Diff against 4 commits back (269 changed/3609+,6898-)	0.25	3.99	16x
Diff Tags	Diff two tags against each other (v1.9.1.0/v1.9.3.0)	1.17	83.57	71x
Log (50)	Log of the last 50 commits (19 kB of output)	0.01	0.38	31x
Log (All)	Log of all commits (26,056 commits – 9.4 MB of output)	0.52	169.20	325x
Log (File)	Log of the history of a single file (array.c – 483 revs)	0.60	82.84	138x
Update	Pull of Commit A scenario (113 files changed, 2164+, 2259-)	0.90	2.82	3x
Blame	Line annotation of a single file (array.c)	1.91	3.04	1x

Figure 1.2: Speed comparison between Git and Subversion for several operations in both CVS. The time needed for the operation is given in seconds. The last column is the factor by which Git is faster. The comparison is taken from [4].

Killerfeature: Branching

But the feature what makes Git so advanced to other VCS is how branching is implemented. A branch is a divergence from the main line of the development. This is helpful to test new code without messing with the entire project. If the change works the branch can be merged into the main branch, the master branch. The implementation of branches in Git takes advantages of the fact that Git saves an update of the repository as a snapshot of the entire repository. A branch is implemented as a pointer to one specific state of the repository. Because a branch is only a pointer makes it extremely lightweight and creating or switching between branches it nearly instantaneous. This encourages the creation of branches multiple times during one working step for every size of working task. That no contributor is changing the original repository everyone is synchronizing with before the changes are stable on the local master branch is very convenient and safeguarding.

1.3 Working with branches in Git

Figures 1.3 - 1.5 illustrate how branching makes work easier. The snapshots of the history C0 to C2 (=commit) are completely linear without any branches. Afterwards some work were done on an issue (iss53) for which a branch and a change were made (C3). But before the issue could be solved an urgent problem was found on the master branch. Therefore another branch called hotfix was made and some correction are done on this branch to solve the problem (C4).

When the changes on the hotfix branch solve the problem, the hotfix branch can be merged into the master branch. In this case the master branch can reach the hotfix branch by just following the history without divergences. This merge is called fast-forward merge and is done by Git automatically. The pointer of the master branch moves forward to commit C4 and the hotfix branch gets removed (see figure 1.4.)

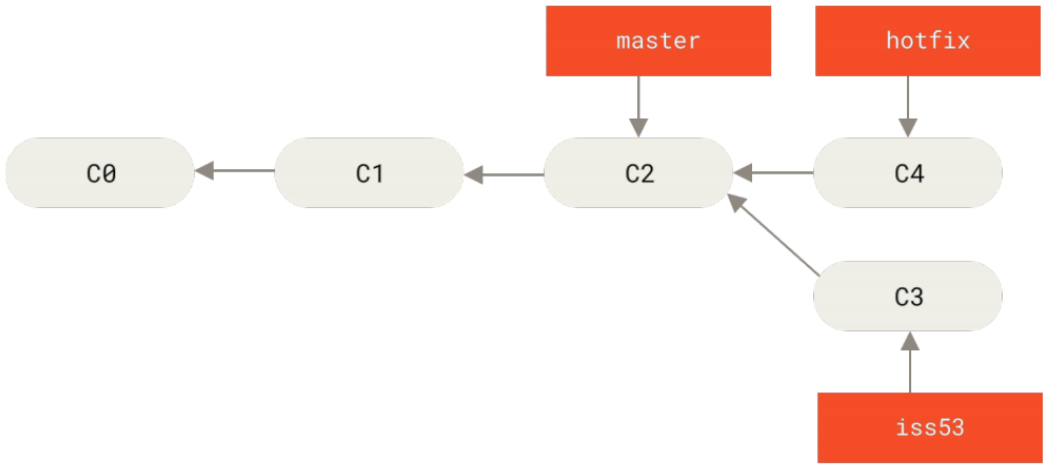

Figure 1.3: Example of a history of a repository history (C0 to C4) with two branches (orange pointers to points in the history). Figure is taken from [1].

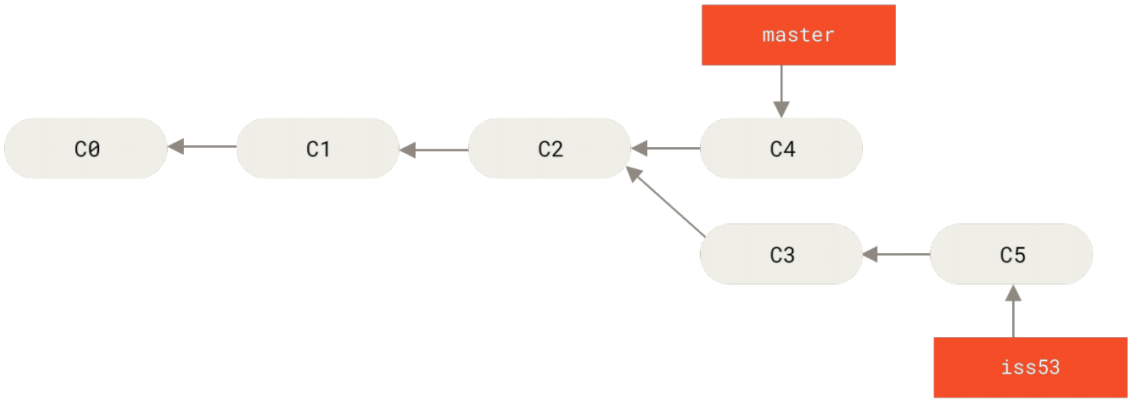

Figure 1.4: Sequel of the example in figure 1.3: "Fast-Forward" merge. Taken from [1].

Afterwards the work on branch iss53 is continued and solved with another commit (C5) and is ready to be merged into the master branch. But this time the two branches are diverged and there is no straight connection between the two branches. As long as different files were changed on the two branches Git can make the merge automatically with the recursive strategy: Git goes back to the last commit the two branches had in common and starts there to merge the changes into. Afterwards a new commit is created with two commits as parents (C6, see figure 1.5). The branch issue53 can be removed.

Because creating, removing and switching between branches is nearly instantaneous, this kind of workflow is supported by Git. If the same file is changed in two diverged branches Git raises a conflict when it is tried to merge them together. The merge has to be done manually to decide which version to keep. But Git offers several tools for a manual merge.

There exists two common workflows with Git branches: longrunning and topic branches.

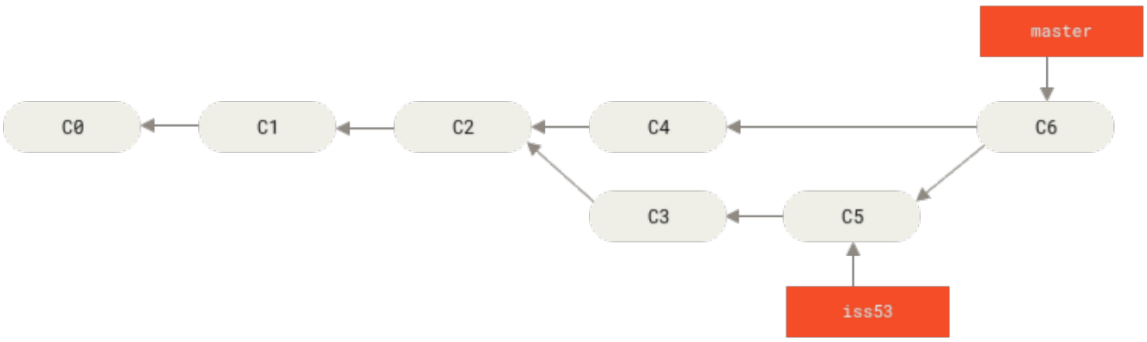

Figure 1.5: Second sequel of the example shown in figure 1.3: "Merge made by recursive strategy". Taken from [1].

Longrunning branches stands for branches that are used as long as they are not completely stable. There could be several branches of different levels of stability. The topic branches exists only short. They are created for one certain task, e.g. the hotfix in figure 1.3, are merged into the master branch and afterwards removed. With this workflow branches are created and removed several times a day.

2 Collaboration on one project

Collaboration is necessary when working on a project. Nowadays no one is working isolated on a topic without communication and sharing data with others. It is vital that every collaborator has access to the data and everyone regularly synchronizes with the work of the others to maintain a seamless workflow. This can be implemented by a remote repository.

2.1 Remote Repository

Remote repositories are hosted somewhere else than the local machine. They are necessary for collaboration with others because the main repository, also called origin repository, has to be accessible to all which is not the case for local machines. At the beginning of the projects everyone clones the remote repository on its local machine. When the changes are stable and merged into the local master branch, they are ready to get pushed to the remote repository. But before that it has to be checked if changes to the origin repository have been made and if this is the case, the changes have to be pull and merged into the local master branch to avoid correlations with other changes. Afterwards the updated master branch can be pushed to the remote repository and the collaborators can synchronize to it. The synchronization with the other collaborators is necessary and not always a straight forward task. Good communication is mandatory.

Hosting Git

To host a Git remote repository it has to be accessible for everyone. Four protocols to transfer data are available for Git: local, HTTPS, SSH and git itself.
A local protocol requires a shared filesystem for all collaborators which might be the case for companies. This has the advantage that the access already exists and it is easy to get code from collaborators but it is generally difficult to set up and the chance of accidental damage exists.
The HTTP protocol can use various HTTP authentication mechanisms and encryption but can be difficult to set up. Due to its high security options it is the most common protocol used, especially for open source projects.
SSH protocols are commonly used when the server is hosted by the company because the ssh connection already exists and it is safe. But due to the fact that collaborators needs ssh access to the machine it is unsuited for open access projects.
The Git protocol has the fastest transfer of all methods but has no authentication. It is often used in combination with another protocol which then is used for writing and Git

is used for read-only access.
Next to buying or renting a server for hosting a Git repository, there are also several open-source solutions. Some are specifically set for Git and a list can be found under [5]. They differ in the amount of space for the repositories, how many collaborators per repository and the number of free private and open-source repositories. Examples for server host that support also other VCS are bitbucket.org (Mercurial) and Visual Studio Team Services (TFVS), while GitLab supports only Git.

2.2 GitHub

Figure 2.1: GitHub logo taken from [6].

But the largest host for Git repositories is GitHub. It is free to use with unlimited private and open source repositories, but with a maximum of three collaborators for a private repository. Next to the general hosting, GitHub can be used for issue tracking, code review and many more tasks. The paid services and features of GitHub includes advanced tools and an increased limit of collaborators as well as memory space. What makes GitHub so popular is opportunity to simplify communication and comments with flavoured markdowns like links, quotes, task lists, code snippets and even emotes. There also exists a lot of introduction material for complete beginners [7],[8],[9]. For example, how to move a local repository on GitHub to enable collaboration on an already existing project [10]. Futhermore, GitHub is also attractive for experienced programmers and large companies because the GitHub base is hackable to personalize the server or to integrate external services.
Despite all these advantages for all kind of skilled programmers, it has to be mentioned that the owner of the origin repository removes the account, all hosted repositories by that account get removed.

GitHub Workflow of an Open-Source Repository

For open-source projects everyone on GitHub can see the repository, but only a few people have write access, which is necessary to insure quality code. But other programmers who wants to contribute can do that by cloning the repository into their GitHub account, which is called "Fork a project". This is how a workflow would look like:

1. Fork the project

2. Create a topic branch
3. Make some commits to improve the projects
4. Push branch to own GitHub projects
5. Open a Pull Request on GitHub
6. Discuss and optionally continue to write commits
7. Project owner merges or closes Pull Request
8. Synchronize the updated master branch back to own fork

Therefore everyone who wants can contribute but in the end the owner of the projects decides what is helpful and gets included.

Final Remark

This is the report to my presentation about the VCS Git in the seminar "Softwareentwicklung in der Wissenschaft". It will held only a part of the information and visualisation I presented in the seminar, therefore I recommend to look at the presentation as well. Especially, because it was a presentation mostly done as an introduction to Git and version control in general. In the presentation I show how to set up a first Git repository and how to connect it to a GitHub server. The presentation can be found under `https://wr.informatik.uni-hamburg.de/teaching/sommersemester_2020/softwareentwicklung_in_der_wissenschaft`.

References

1. Pro Git by Chacon and Staub, The Expert Voice; 2nd Edition
2. https://biz30.timedoctor.com/git-mecurial-and-cvs-comparison-of-svn-software/
3. https://insights.stackoverflow.com/survey/2018 [25.04.2020]
4. https://git-scm.com/
5. https://git.wiki.kernel.org/index.php/GitHosting [01.05.2020]
6. https://github.com/logos
7. https://guides.github.com/
8. https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners
9. https://help.github.com/en/github/getting-started-with-github/git-and-github-learning-resources
10. https://help.github.com/en/github/importing-your-projects-to-github/adding-an-existing-project-to-github-using-the-command-line