Version Control and Collaboration with Git & GitHub

Carson McKee & Artiom Rumiancev

May 9, 2024

Table of Contents

Schedule

- Introduction to Version Control and Git/GitHub.
- Set up and practice using git/GitHub (practical).
- Creating a personal website with Quarto and GitHub Pages.

Table of Contents

What is Version Control?

- Imagine if each time you modified a file, you re-saved it under a new name. E.g. v1.1, v1.2, ...
- This is essentially what version control is - although that particular example would be a very inefficient system.
- More precisely, version control is a system for tracking and managing changes to software.

What is Version Control?

- Imagine if each time you modified a file, you re-saved it under a new name. E.g. v1.1, v1.2, ...
- This is essentially what version control is - although that particular example would be a very inefficient system.
- More precisely, version control is a system for tracking and managing changes to software.
- There are two main ways in which of version control is used:

What is Version Control?

- Imagine if each time you modified a file, you re-saved it under a new name. E.g. v1.1, v1.2, ...
- This is essentially what version control is - although that particular example would be a very inefficient system.
- More precisely, version control is a system for tracking and managing changes to software.
- There are two main ways in which of version control is used:
    - Locally - When you are working alone and just want to track/save your changes.

What is Version Control?

- Imagine if each time you modified a file, you re-saved it under a new name. E.g. v1.1, v1.2, ...
- This is essentially what version control is - although that particular example would be a very inefficient system.
- More precisely, version control is a system for tracking and managing changes to software.
- There are two main ways in which of version control is used:
  - Locally - When you are working alone and just want to track/save your changes.
  - Remotely - When you are working with others and need to share code.

Why use Version Control?

When working alone (local VC) it is useful because:

- It allows you to experiment with working code more freely. If you end up breaking it, you can just revert back to the working version.
- If looking back at old code, you have a paper trail for what each change does.

Why use Version Control?

When working alone (local VC) it is useful because:

- It allows you to experiment with working code more freely. If you end up breaking it, you can just revert back to the working version.
- If looking back at old code, you have a paper trail for what each change does.

When working with others (distributed VC):

- It protects you from losing files to a computer crash etc.
- It allows you to see the changes others have made - and to incorporate them with your changes more easily.
- It prevents multiple people creating version conflicts.
- It allows you to work in parallel/independently to the main work (usually called 'branching'). This is useful for writing experimental code which may break easily.

Improving Reproducibility

- Collberg et al. (2016) were only able to reproduce $\approx 25\%$ of 600 computer science studies they reviewed.

Improving Reproducibility

- Collberg et al. (2016) were only able to reproduce $\approx 25\%$ of 600 computer science studies they reviewed.
- Biometrical Journal tracked reasons for lack of reproducibility in submissions.

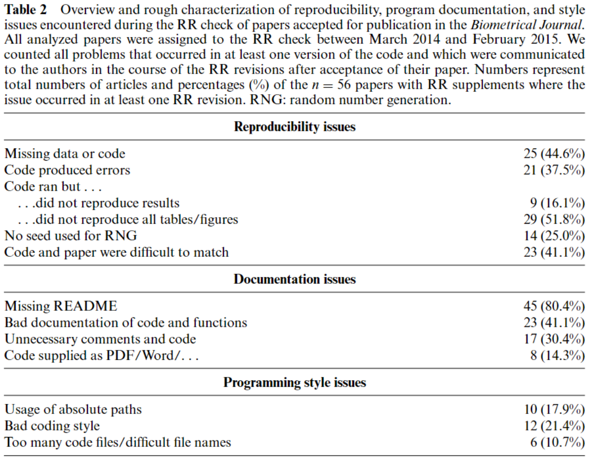


Table 2 Overview and rough characterization of reproducibility, program documentation, and style issues encountered during the RR check of papers accepted for publication in the *Biometrical Journal*. All analyzed papers were assigned to the RR check between March 2014 and February 2015. We counted all problems that occurred in at least one version of the code and which were communicated to the authors in the course of the RR revisions after acceptance of their paper. Numbers represent total numbers of articles and percentages (%) of the $n = 56$ papers with RR supplements where the issue occurred in at least one RR revision. RNG: random number generation.

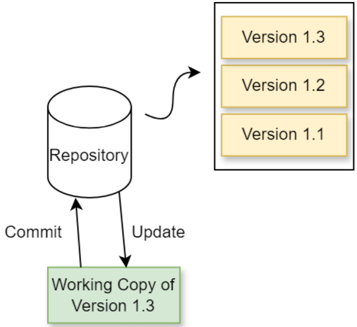
Reproducibility issues	
Missing data or code	25 (44.6%)
Code produced errors	21 (37.5%)
Code ran but . . .	
. . .did not reproduce results	9 (16.1%)
. . .did not reproduce all tables/figures	29 (51.8%)
No seed used for RNG	14 (25.0%)
Code and paper were difficult to match	23 (41.1%)
Documentation issues	
Missing README	45 (80.4%)
Bad documentation of code and functions	23 (41.1%)
Unnecessary comments and code	17 (30.4%)
Code supplied as PDF/Word/. . .	8 (14.3%)
Programming style issues	
Usage of absolute paths	10 (17.9%)
Bad coding style	12 (21.4%)
Too many code files/difficult file names	6 (10.7%)

Improving Reproducibility

- Many journals (such as RSS) will require that any code and data required to reproduce results is openly available.
- Iteratively writing your code using a VC system means that your code is already documented and available for others to use.
- Of course, using VC does not solve all the issues relating to reproducibility, however, if used correctly it can save you some time when it comes to making it publicly available.

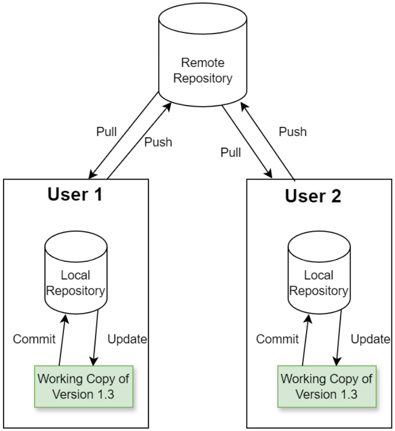
Local Version Control

- Local VC consists of a **repository** and a **working copy**.
- The repository is a database containing all file versions.
- The working copy is where you make edits to the file.
- When you are finished editing, you make a **commit**.
- A commit creates a new version in the repository containing your edits.



Distributed Version Control

- When working with others, we often instead use *distributed* version control.
- Here each user gets both a local repository and a working copy.
- The local repositories are referred to as a **clones**.
- After you commit your changes locally you must also *push* these changes to the remote repo before others can also pick them up.



Simultaneous Editing

- Distributed VC raises the question of how to deal with multiple developers editing the same file at the same time.

Simultaneous Editing

- Distributed VC raises the question of how to deal with multiple developers editing the same file at the same time.
- Simple version control systems may use *file locking*, which means only one developer can modify a file at a given time.

Simultaneous Editing

- Distributed VC raises the question of how to deal with multiple developers editing the same file at the same time.
- Simple version control systems may use *file locking*, which means only one developer can modify a file at a given time.
- However, most VC systems will instead use *merging* (more on this later).

Simultaneous Editing

- Distributed VC raises the question of how to deal with multiple developers editing the same file at the same time.
- Simple version control systems may use *file locking*, which means only one developer can modify a file at a given time.
- However, most VC systems will instead use *merging* (more on this later).
- In these systems, whoever pushes their commits first will create the newest version.

Simultaneous Editing

- Distributed VC raises the question of how to deal with multiple developers editing the same file at the same time.
- Simple version control systems may use *file locking*, which means only one developer can modify a file at a given time.
- However, most VC systems will instead use *merging* (more on this later).
- In these systems, whoever pushes their commits first will create the newest version.
- If someone else tries to push their commits based on previous versions, the system will tell them to first *merge* these changes with the newest commits.

Table of Contents

Git

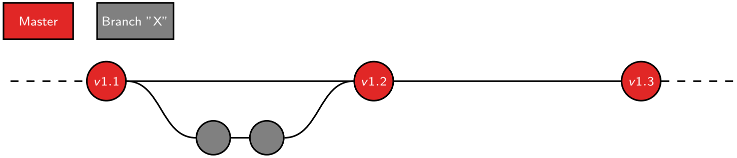
Git is a *distributed* version control system (VCS) that tracks changes to files and directories over time.

- It treats the changes as a series of **snapshots** of the project.
- Users have access to a complete copy and the history of the project.
- As such, they can work offline, make changes, commit the code locally and push the changes to be integrated into the remote repository, where the project is located.
- This fosters collaboration and allows to efficiently track changes, manage code bases, and maintain a complete history of modifications to projects.

Basic concepts

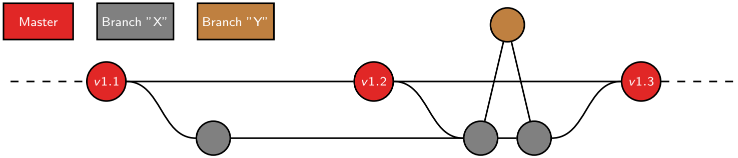
- **Branch** - is a lightweight pointer to a snapshot of the project. The default branch is usually `master`.
- **Commit** - represents a snapshot of the project at a particular point in time. It includes information about the author, a commit message and a unique identifier.
- **Merge** - combines content from multiple branches into one. It integrates changes from the *source branch* to the *target branch* (e.g. from `source` $\rightarrow$ `master`).

Workflow



- **Clone:** To begin working on a project, you typically clone the content of an existing repository (`git clone`) or initialize a new project (`git init`).
- **Add and Commit:** After making changes or adding new features, you stage (`git add`) and commit them to a project (`git commit`). Commits should be frequent and descriptive.
- **Push:** The changes are then integrated into the project (`git push` or `git push --set-upstream` for new branches) located in a remote repository.

Workflow - Continued



- **Pull:** To keep your local repository up to date, you typically fetch (`git pull`) the changes made to the repository by your collaborators.
- **Stash and Pop:** At some point, you may need to store (`git stash`) your changes temporarily, before you update your local repository, and later continue working on them (`git stash pop`).
- **Branching and Merging:** You can also create a new branch (`git branch`) and begin working on it (`git checkout`). After the changes are made, the branch can be merged with an existing one (`git merge`).

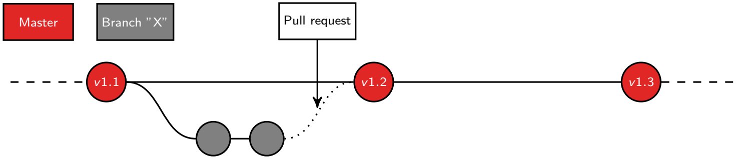
Table of Contents

GitHub

GitHub is a *cloud-based service* for hosting remote repositories (repos).

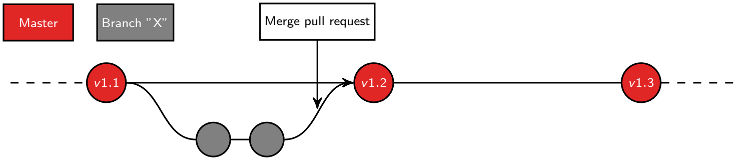
- Thus, when combined with using Git locally on your machine, it forms a distributed version control system that can be used by multiple collaborators.
- GitHub provides additional features such as issue tracking, project management, continuous integration and code review.

Pull Requests



- **Pull request**: A way to propose changes to the project on host platforms such as GitHub or GitLab. Pull requests are native to Git (`git request-pull`).
- They enable contributors to propose, review, and merge changes in a transparent and collaborative manner.
- Once a pull request is open, the potential changes can be reviewed and follow-up commits can be added before the changes are integrated into the target branch.

Merging Pull Requests



- By default, pull requests can be made at any time, unless there are conflicts between the source and target branch.
- These conflicts need to be addressed manually, in order to mitigate merge errors.
- Furthermore, additional checks/tests (if any) need to be passing before the source branch can be merged with the target branch.

What to do now

- We've provided two sets of instructions for setting up and practising using Git/GitHub using either Python (with VS Code) or R (with RStudio).
- You'll find these instructions on the public GitHub repo 'artrumiancev/intro_to_github'.

R

Python

References

[1] Christian Collberg and Todd A. Proebsting.
Repeatability in computer systems research.
Communications of the ACM, 59:62–69, 2 2016.

[2] Benjamin Hofner, Matthias Schmid, and Lutz Edler.
Reproducible research in statistics: A review and guidelines for the biometrical journal.
Biometrical journal. Biometrische Zeitschrift, 58:416–427, 3 2016.