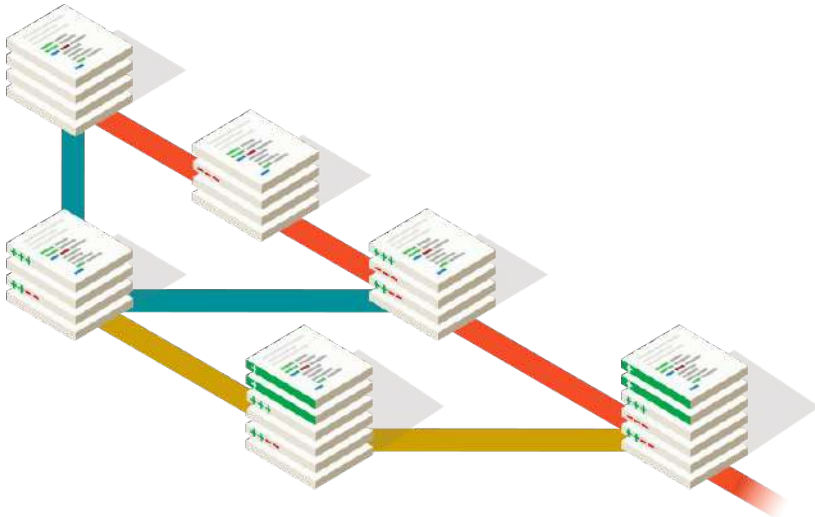


Big Data Ecology

Version Control

Christian König

Ecology and Macroecology Lab
Institute for Biochemistry and Biology
University of Potsdam

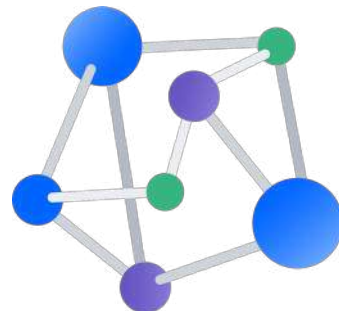



What is version control?

- A system for managing changes to files (usually source code) that:
 - Allows multiple developers to work on the same source files
 - Stores a complete change history of all tracked files
 - Provides tools to safely experiment with your code without breaking it
 - Enables transparent and collaborative code development across the world
- Different architectures (centralized vs. distributed), implementations (e.g.: SubVersion, CVS, Git) and online hosters (e.g.: Bitbucket, GitLab, GitHub)



- Most powerful and widely used version control system available
- Distributed architecture → your local repository is self-contained and includes the full development history
- Elegant branching and merging logic
- Free web hosting with powerful features, e.g. on GitHub or GitLab





git Basics - File organization

1. Working directory / working tree

Your root project folder

2. Staging area

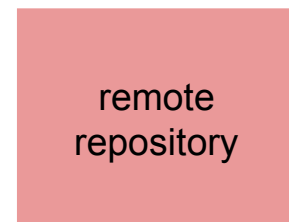
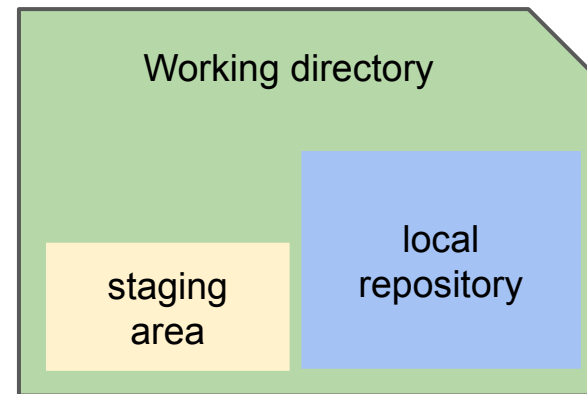
Snapshots of files that want to put under git version control

3. Local repository

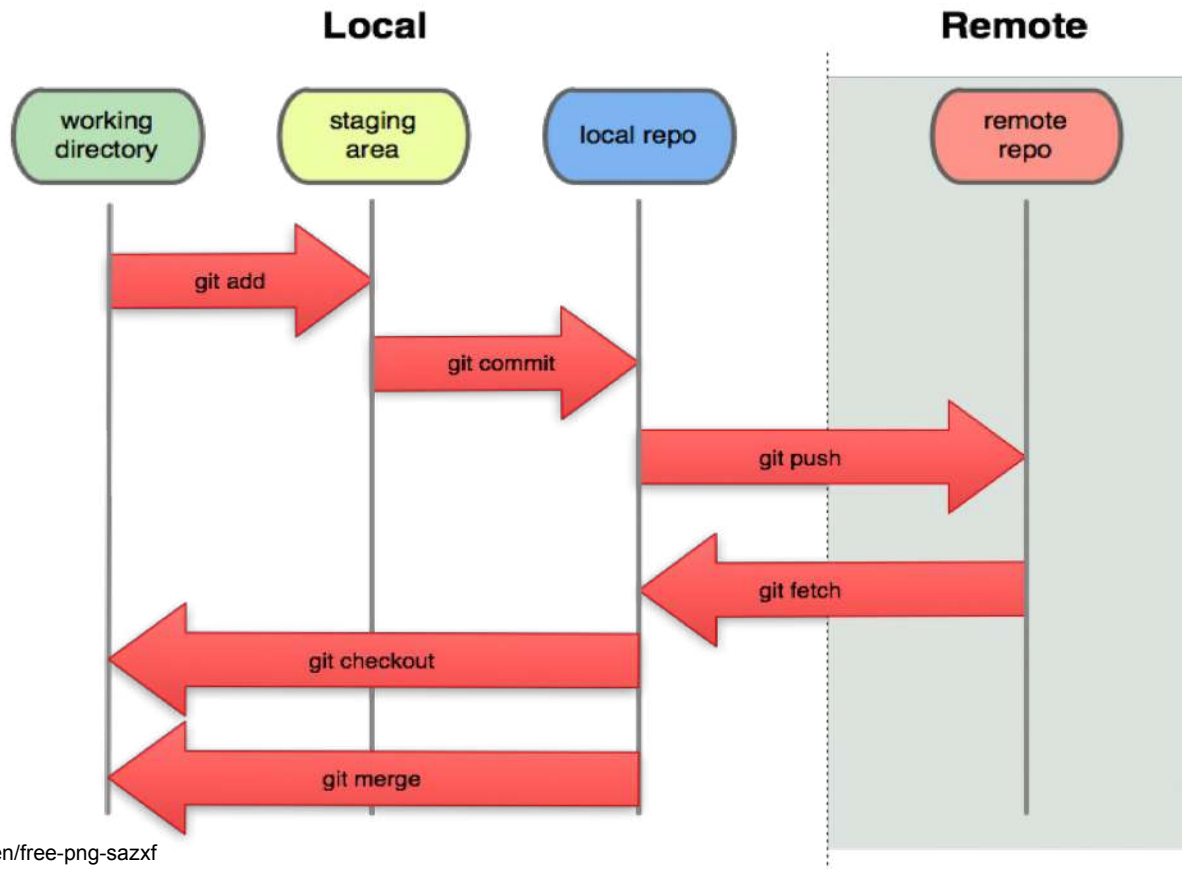
Local files under git version control

4. (Remote repository)

Remote repository of the same project, but with potentially different contents

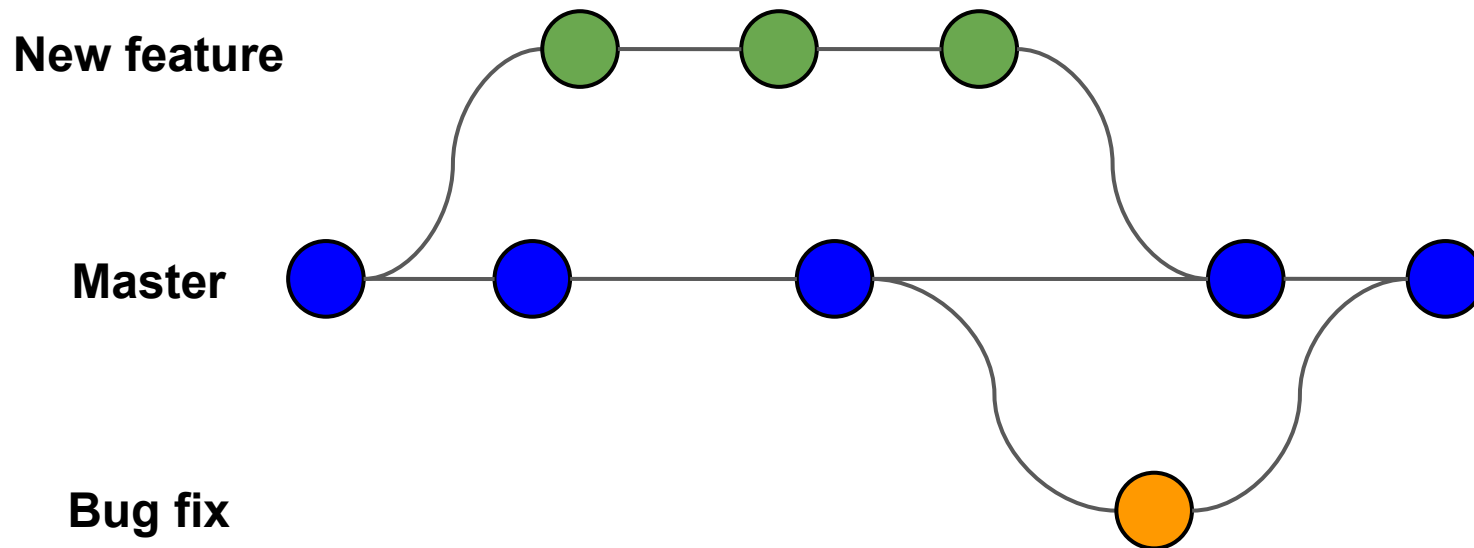


Basics - Workflow overview





Basics - Branching and Merging





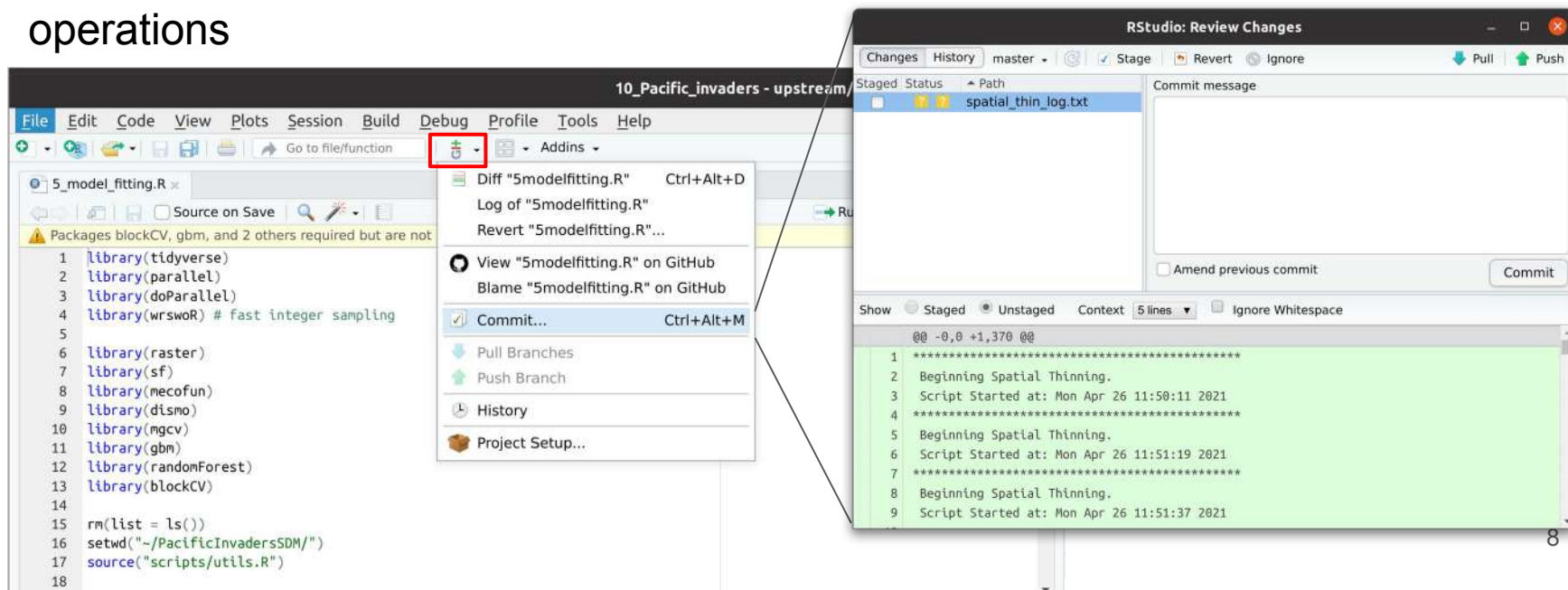
git Web Hosting

- Web platforms can host remote repository
- They offer a number of additional powerful features such as:
 - collaboration features, e.g. issue tracking, forks, pull requests
 - visualization tools, e.g. activity graphs, diffing tools
 - continuous integration, i.e. automated compilation and unit testing of new code
- The most popular git web hosting platform is [Github](#)
- The University of Potsdam runs its own GitLab-based versioning service: [Git.UP](#)



git IDE integration

- Most modern IDEs have integrated Git support
- RStudio has a nice graphical interface for the most common git operations





git Commands

GIT BASICS

<code>git init</code> <code><directory></code>	Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
<code>git clone <repo></code>	Clone repo located at <code><repo></code> onto local machine. Original repo can be located on the local filesystem or on a remote machine via HTTP or SSH.
<code>git config</code> <code>user.name <name></code>	Define author name to be used for all commits in current repo. Devs commonly use <code>--global</code> flag to set config options for current user.
<code>git add</code> <code><directory></code>	Stage all changes in <code><directory></code> for the next commit. Replace <code><directory></code> with a <code><file></code> to change a specific file.
<code>git commit -m</code> " <code><message></code> "	Commit the staged snapshot, but instead of launching a text editor, use <code><message></code> as the commit message.
<code>git status</code>	List which files are staged, unstaged, and untracked.
<code>git log</code>	Display the entire commit history using the default format. For customization see additional options.
<code>git diff</code>	Show unstaged changes between your index and working directory.

UNDOING CHANGES

<code>git revert</code> <code><commit></code>	Create new commit that undoes all of the changes made in <code><commit></code> , then apply it to the current branch.
<code>git reset <file></code>	Remove <code><file></code> from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes.
<code>git clean -n</code>	Shows which files would be removed from working directory. Use the <code>-f</code> flag in place of the <code>-n</code> flag to execute the clean.

REWRITING GIT HISTORY

<code>git commit</code> <code>--amend</code>	Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit's message.
<code>git rebase <base></code>	Rebase the current branch onto <code><base></code> . <code><base></code> can be a commit ID, branch name, a tag, or a relative reference to HEAD.
<code>git reflog</code>	Show a log of changes to the local repository's HEAD. Add <code>--relative-date</code> flag to show date info or <code>--all</code> to show all refs.

GIT BRANCHES

<code>git branch</code>	List all of the branches in your repo. Add a <code><branch></code> argument to create a new branch with the name <code><branch></code> .
<code>git checkout -b</code> <code><branch></code>	Create and check out a new branch named <code><branch></code> . Drop the <code>-b</code> flag to checkout an existing branch.
<code>git merge <branch></code>	Merge <code><branch></code> into the current branch.

REMOTE REPOSITORIES

<code>git remote add</code> <code><name> <url></code>	Create a new connection to a remote repo. After adding a remote, you can use <code><name></code> as a shortcut for <code><url></code> in other commands.
<code>git fetch</code> <code><remote> <branch></code>	Fetches a specific <code><branch></code> , from the repo. Leave off <code><branch></code> to fetch all remote refs.
<code>git pull <remote></code>	Fetch the specified remote's copy of current branch and immediately merge it into the local copy.
<code>git push</code> <code><remote> <branch></code>	Push the branch to <code><remote></code> , along with necessary commits and objects. Creates named branch in the remote repo if it doesn't exist.

Practical

Work through the R practical [version_control.md](#)

Further readings

Tutorials, Videos & Documentation:

<https://git-scm.com/doc>

<https://www.atlassian.com/git>

<https://www.youtube.com/watch?v=2sjqTHE0zok>

Git & RStudio:

<https://support.rstudio.com/hc/en-us/articles/200532077-Version-Control-with-Git-and-SVN>

Github:

<https://guides.github.com/activities/hello-world/>

Troubleshooting:

<https://stackoverflow.com/questions/tagged/git?sort=MostVotes&edited=true>