Big Data Ecology

Version Control

Christian König

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

Source: https://git-scm.com/

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)

git

- 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
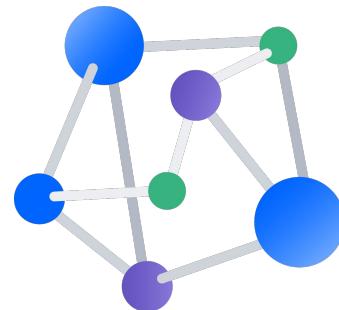
git Basics - File organization

1. **Working directory / working tree**
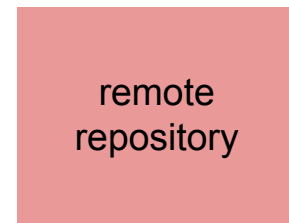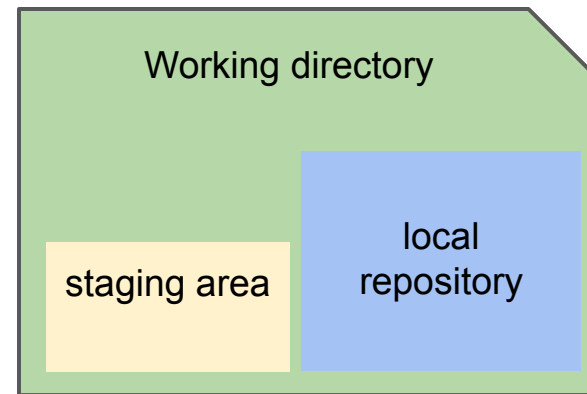
   Your root project folder

2. **Staging area**

   Snapshots of files that want to put under git version control
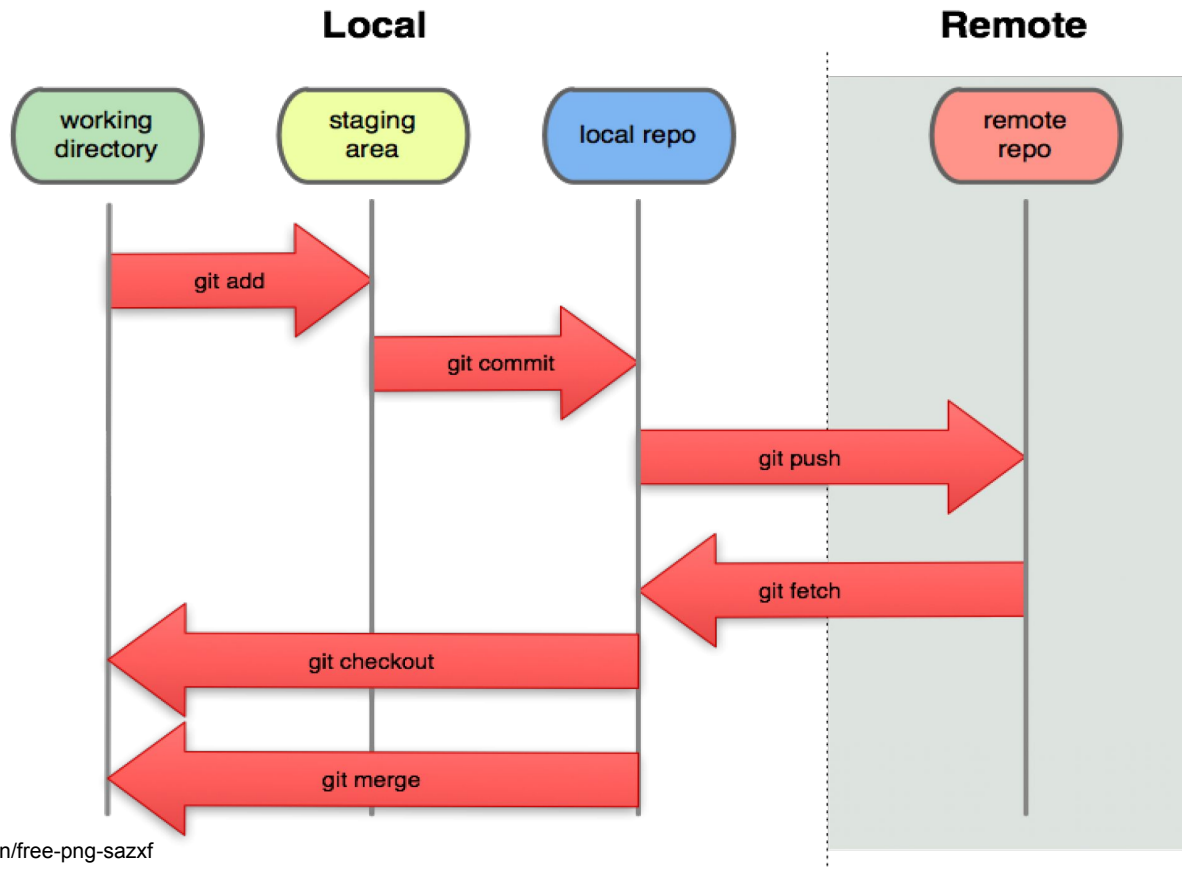
3. **Local repository**

   Local files under git version control
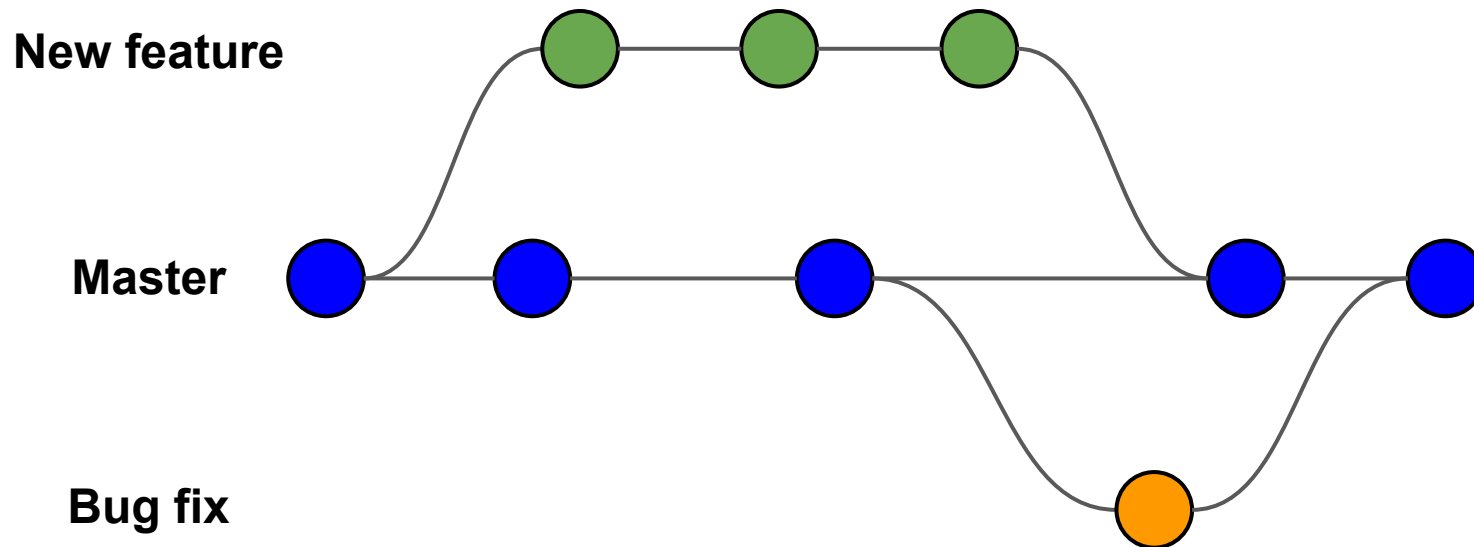
4. **(Remote repository)**

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

4

git Basics - Workflow overview

Source: https://www.pngwing.com/en/free-png-sazxf

git 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
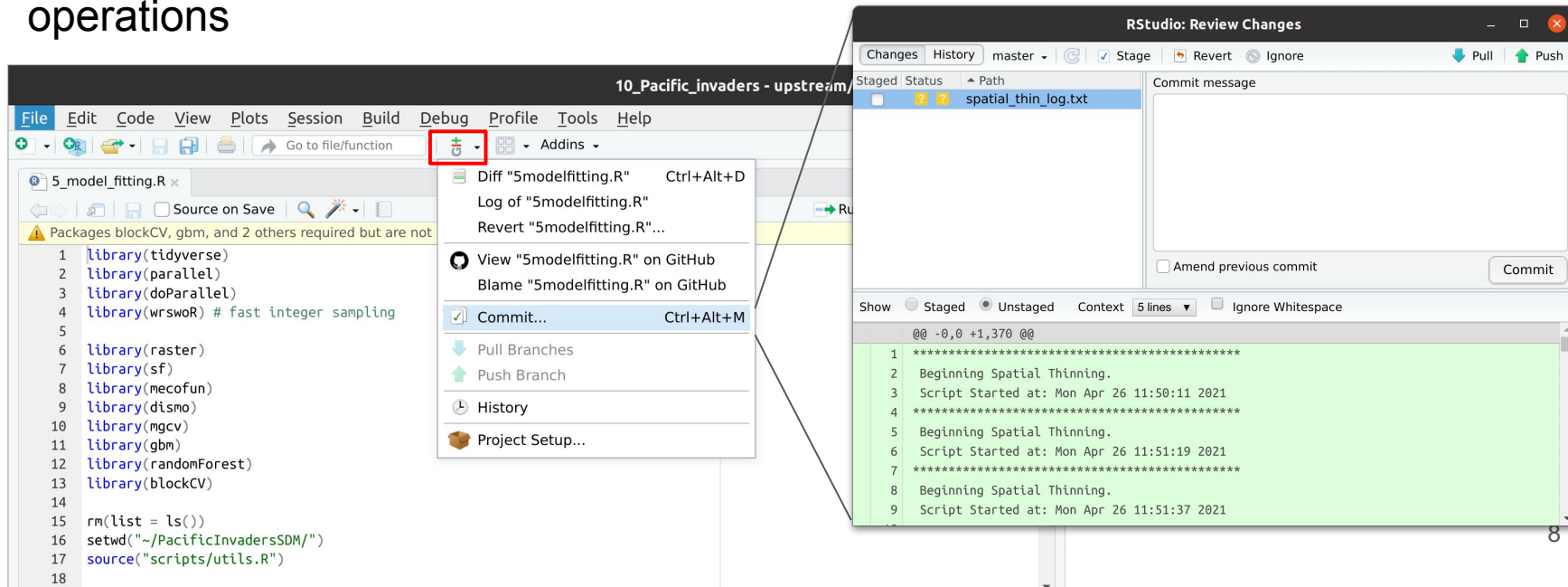
git Commands

GIT BASICS

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

UNDOING CHANGES

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

REWRITING GIT HISTORY

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

GIT BRANCHES

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

REMOTE REPOSITORIES

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

Practical

Work through the R practical <u>version_control.md</u>

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