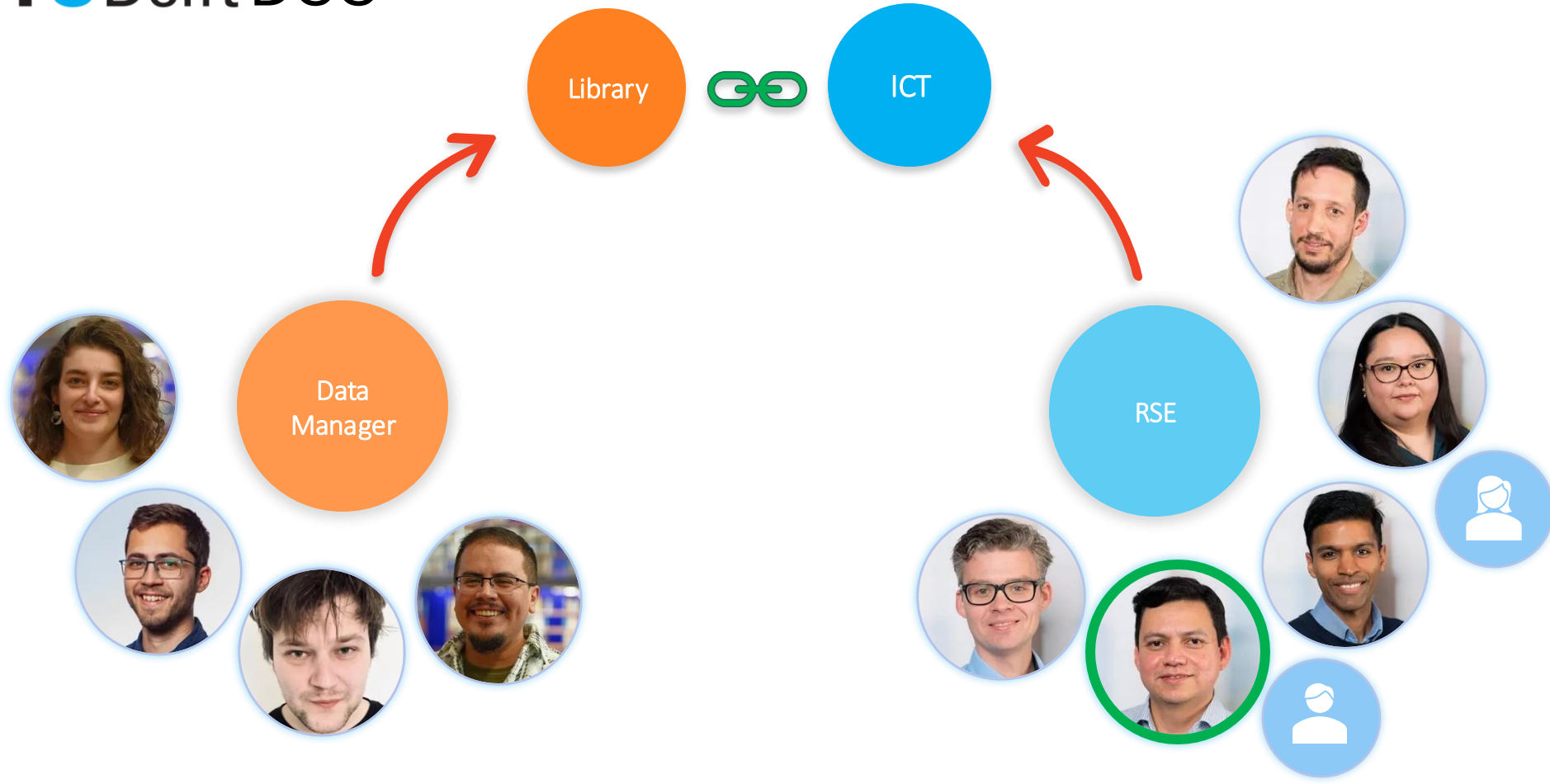




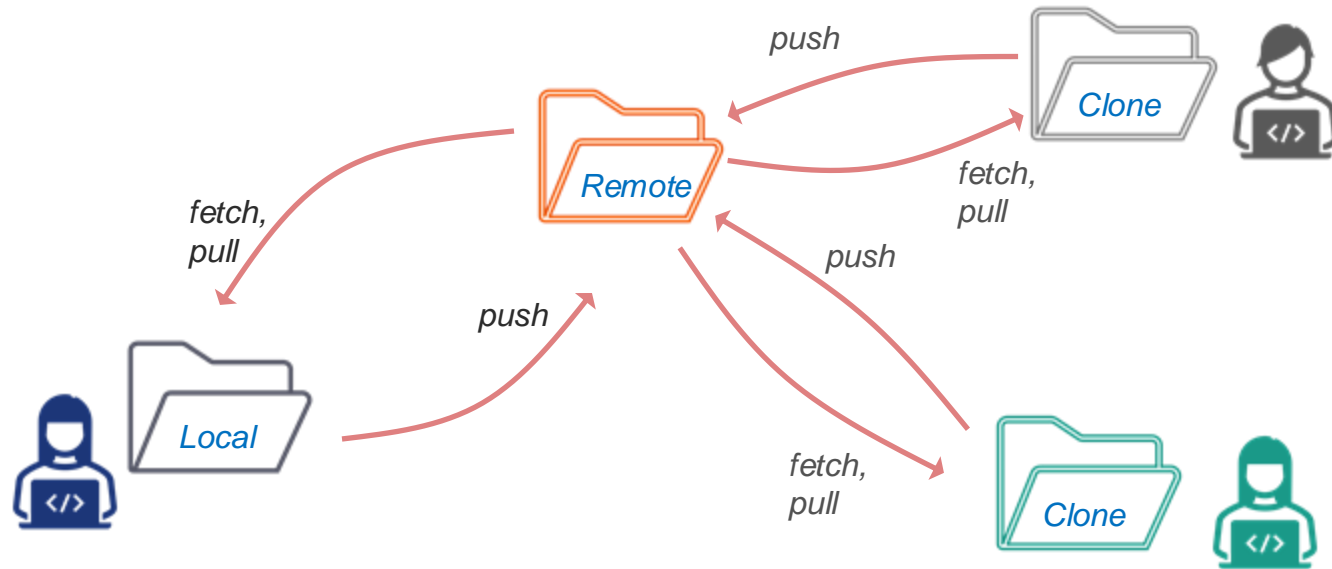
Operations with Remotes

Workshop ‘Version control and collaborative development for research Software’
Delft University of Technology, 07-10 October 2024

A course developed and delivered by Manuel Garcia Alvarez and Giordano Lipari



Operations with Remotes





Remote repositories and operations Summary

- Remotes are very often hosted on the Internet/Network (GitHub, GitLab, BitBucket).
- Remote repositories enable distributed version control and collaboration.
- Local repositories can be associated to one or more remotes.
- ``git clone`` clones remote repository to a local computer
- ``git push`` pushes changes from a local to a remote repository.
- ``git pull`` involves two operations ``git fetch`` and ``git merge``.
- Remote and local repositories do not sync automatically.

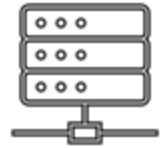
Connecting to remotes in the cloud (GitHub)



Local Client



Local Client



Remote Server



public



Collaborative software development

Lesson 3 of the workshop ‘Version control and collaborative development for research Software’

Delft University of Technology, 07-10 October 2024

A course developed and delivered by Manuel Garcia Alvarez and Giordano Lipari



Motivation

Developing high-quality software requires more than programming and technical skills. Exceptionally good programmers may produce high quality software by themselves. But good programmers will need to collaborate in order to develop complex, high-quality software.




Lesson 3

3|2 Collaborative platforms

3|1 Collaborative development for research software

3|3 Collaborative workflows



Episode

Collaborative Platforms



Exercise 1: Starting with Collaboration [10 min]



1. Clone the Check-in repository via SSH:
<https://github.com/WorkshopGitcodev/check-in>
2. Make a copy of the file ``check-in/template.md`` in the same directory; give it an anonymous name, ex. `<name-initials><2-last-digits-phone>.md`
3. Open **your copy** of ``template.md`` and add something to the lists in the file.
4. Commit your changes and push them to the remote repository. You might experience difficulties doing that, follow the suggestions given by Git.
5. Reflect on the difficulties you faced, and how you could avoid them.



Episode

Collaborative Development for research



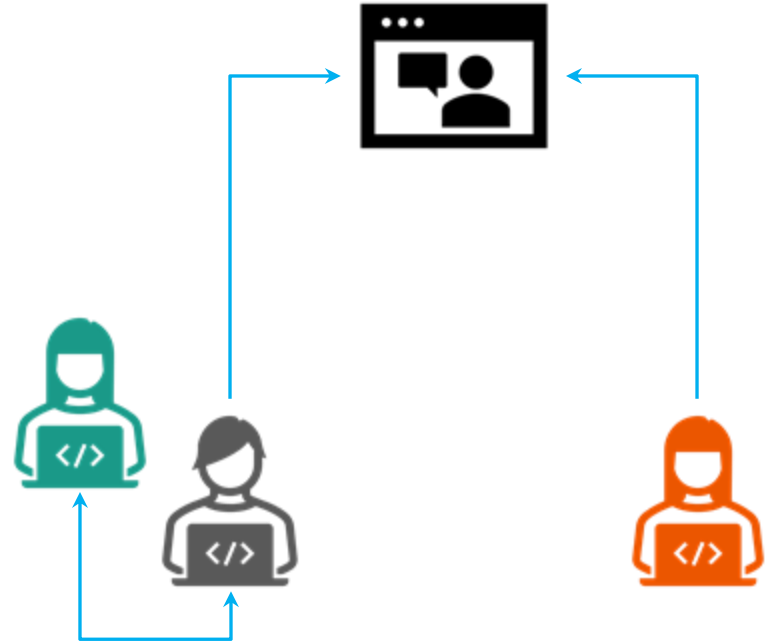
Software Development

The process of designing, coding, testing, and maintaining software systems. It involves a series of activities to create, deploy, and manage software to meet specific requirements and solve a particular problem.

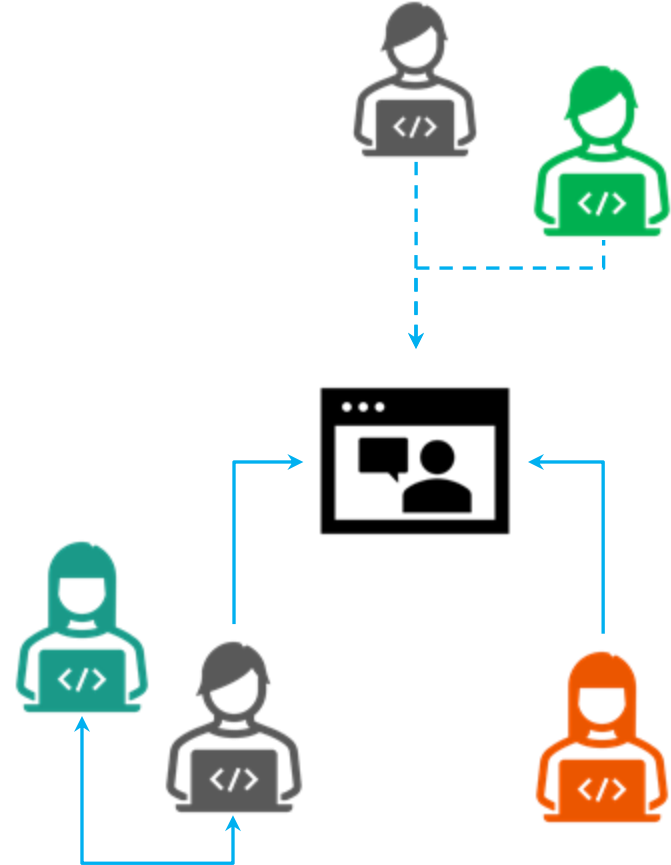
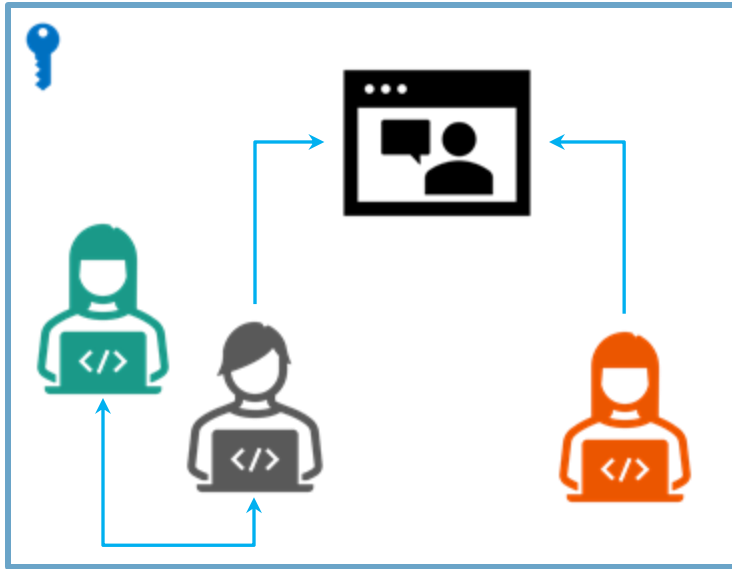


Collaborative Software Development

It involves multiple developers working together to create software products. It encompasses various practices, tools, and techniques to enhance teamwork and productivity.



Open and Closed Collaboration



Some considerations on open collaboration



- Adds recognition to a research project.
- Others may contribute by adding features, testing, debugging and solving issues. This may result in better-quality software.
- Supports building a community of users and collaborators right from the start.
- Adds transparency to research software which increases the reproducibility of research outputs.
- Managing collaborations becomes more challenging and time-consuming as the number of active collaborators grows.
- Others may steal ideas and benefit economically at the cost of somebody's hard work.
- Quality and reliability may be compromised.
- May lead to fragmentation of efforts and disputes.

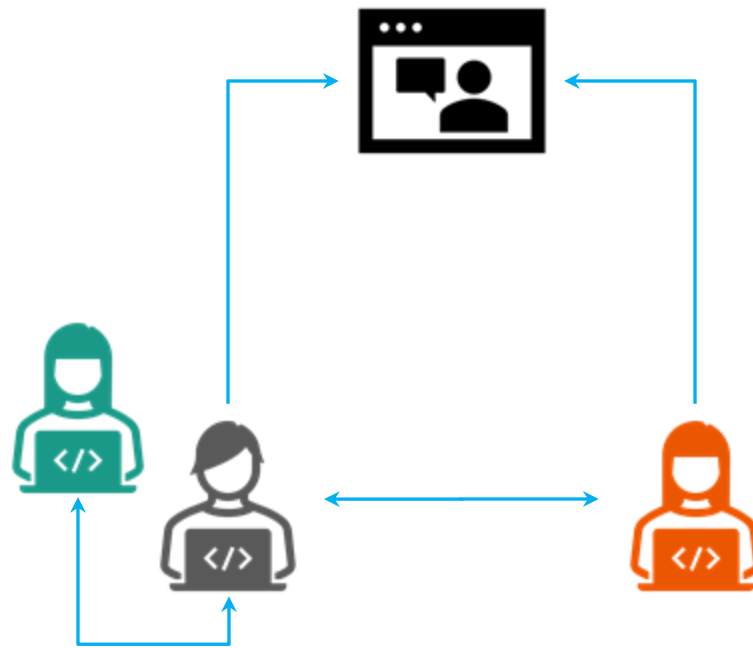
Open or Closed Collaboration?



“As open as possible, as closed as necessary”

When to aim for a collaborative development

- Is the development or delivery of software an important output for a research project?
- How many people are involved in the research project?
- How many will be actively involved in writing software?
- Is it worth? Do the benefits outrun the costs? Is it effective and efficient for a research project?



Questions?

Research software project management

A **research software project** is a research project or part of a research project whose objectives include developing software for a **well-defined purpose**.

Key factors to consider:



purpose



people

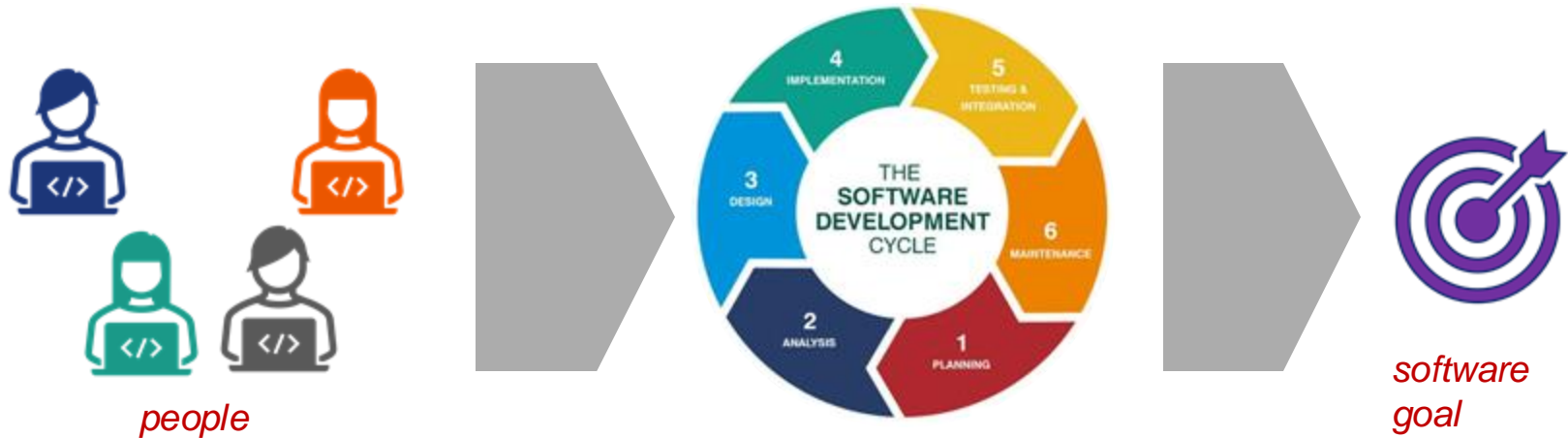


time







maintenance

An approach to organise research teams for collaborative development of research software*



Roles and responsibilities*

ROLE	RESPONSIBILITIES
	<i>Project owner</i> Defines and clarifies the tasks of the development team.
	<i>Administrator</i> Assist the development team in achieving the tasks, manage access to the project resources and often is responsible for maintenance.
	<i>Reviewer</i> Review code from collaborators based on coding style guidelines and quality checklists. Usually someone with a good understanding of the purpose and requirements.
	<i>Collaborator</i> Develops the software (design, implementation, testing, etc.)

Questions?

(Group) Exercise 2. Roles and Responsibilities [6 min]



1. Make teams of 3 to 4 people. You will work together in the upcoming group exercises.
2. Assign roles and responsibilities to each member; you should end up with:
 - **1 project owner**
 - **1 administrator**
 - **1 or more collaborators**
 - **1 or more reviewers**
3. Give a *memorable name* to your team.



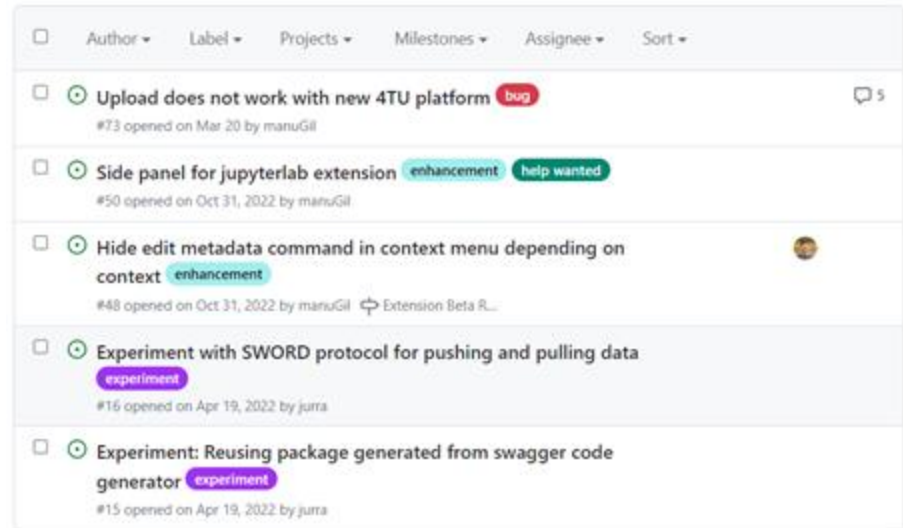
Episode

Collaborative Workflows



Issues

Document and track ideas and tasks in a development project. They're facilitate **planning, discussing and tracking the progress** of a software project.

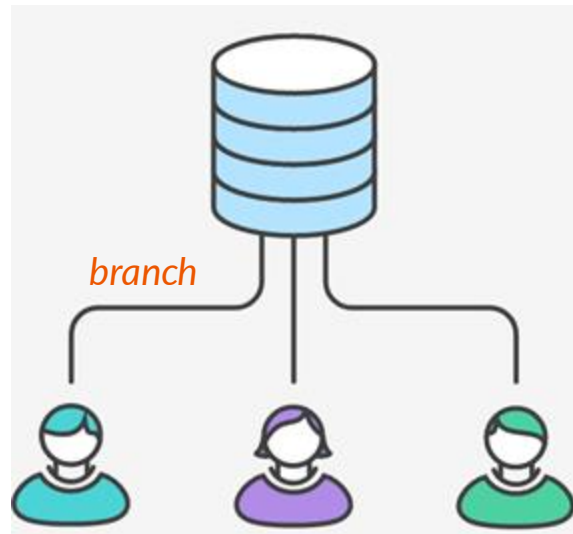


Documenting Issue: Best Practices

- Use issues to **plan and document** and monitor the **progress** of the development
- Let **collaborators know what and how** you would like to use issues (Collaborative guidelines)
- Avoid **duplication**.
- Use **templates** if you need to structure the content of issues.
- When reporting issues, be **clear** and as **specific** as possible.
- If reporting bugs or errors, **provide details and examples** for reproducing them.
- **Labels** will help to filter issues by topic. But do not overuse them.
- **Vulnerability reports** should be reported elsewhere.
- Do not underestimate the **time** it will take to **manage issues** in very active projects.

Workflows

Centralised Workflow. Use a **branching model** to manage contributions.



(Team) Exercise 3. Branching workflow [12 min]

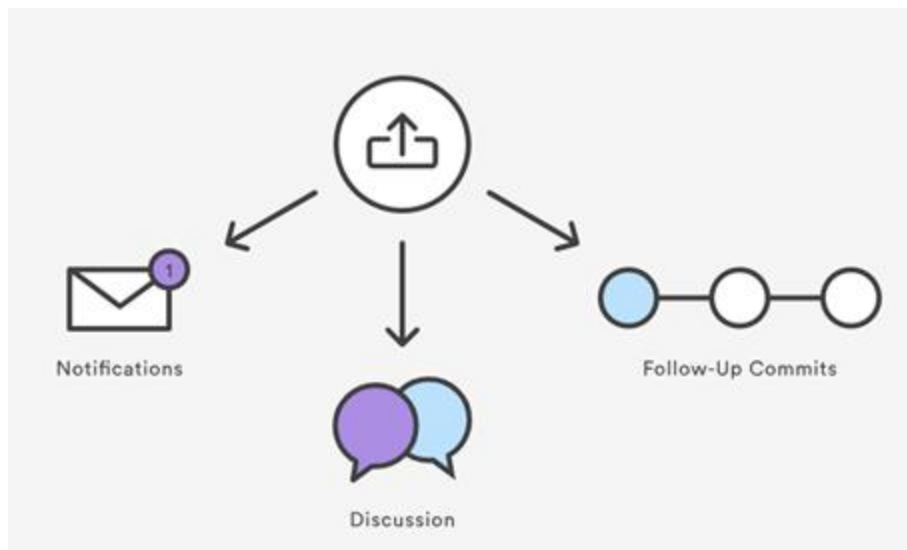


Working as a team implement the branching workflow to work on the following tasks:

1. [Administrator] creates a repository for the team using this template:
<https://github.com/WorkshopGitcodev/collab-branching>
2. [Administrator] invites all team members to the team's repository as collaborators.
3. [Team] reads the *TODO.md* file and each member choses a task for the next step.
4. [Collaborator] each member opens an issue for the chosen task.
5. [Collaborator] applies the branching model to complete the chosen task.
6. [Collaborator] commits and pushes changes to the team's repository.

Pull requests

Let others know someone has made changes to a branch, so that they can be discussed and reviewed.



(Team) Exercise 4. Pull requests [6 min]



Working as a team merge the changes made in the previous exercise into the main branch of the team's repository.

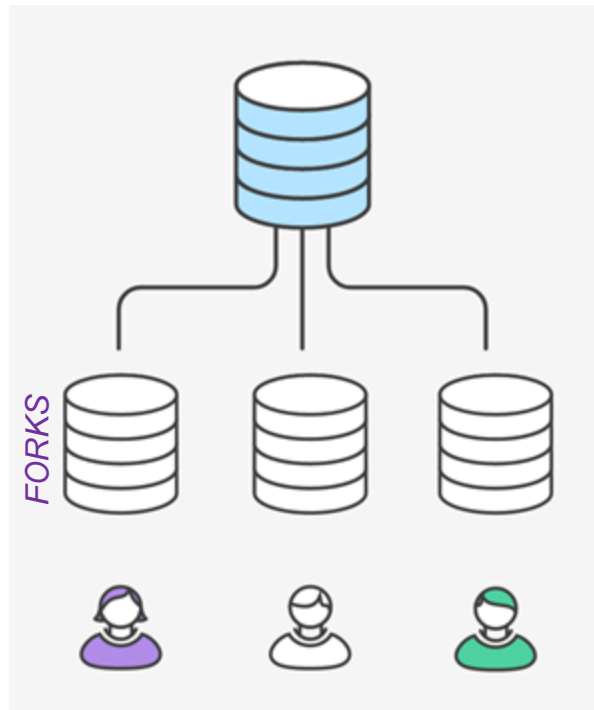
1. [Collaborator] create a pull request for their own branch.
 - Give your pull request a meaningful name, and a short description.
2. [Collaborator] Merge the pull request to the main branch using the method of their choice.
3. Are there any conflicts? Resolve them using the GitHub GUI. Ask for help if you need to.
4. Check the main branch to confirm that your changes have been merged.

Workflows

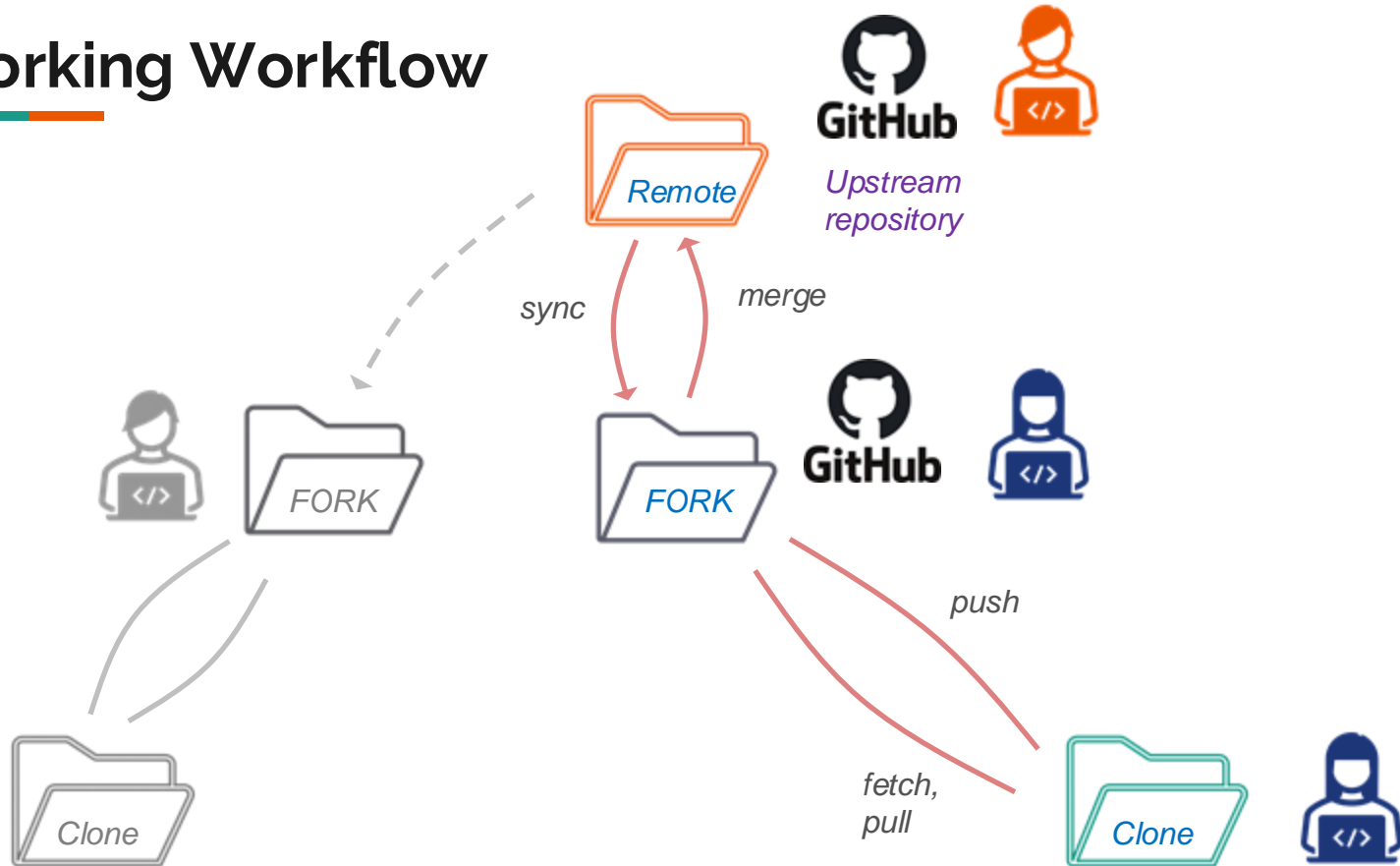
Decentralized Workflow. Use a **forking and branching model** to manage contributions

- Helps an administrator of a project to open the repository to contributions from any developer without having to manually manage authorization settings for each individual collaborator.
- Gives collaborators freedom to make major changes to the project before proposing them (creating pull requests).

More on git workflow



Forking Workflow



(Team) Exercise 3. Decentralised workflow [15 min]




Working in teams, apply FAIR principle to a Git repository using a FAIR software checklist

1. [Administrator] creates a base repository for the team using the *collab-forks template*: <https://github.com/WorkshopGitcodev/collab-forks> For the name of the repository use `<team-name>-base`
2. [Team] Go thru the list in the `TODO.md` and assign a task to each member.
3. [Collaborator] open an issue about the item you chose in the base repository.
4. [Collaborator] fork the team's base repository to their accounts.
5. [Collaborator] Clone your fork to your local machine, make changes to address the issue and push changes to your fork.
6. [Collaborator] Make a pull request from your fork to the upstream repository on the main branch. Merge the changes if there are not conflicts.

Questions?

Summary



Working in teams, apply FAIR principle to a Git repository using a FAIR software checklist

- **SSH** is a secure way to connect to Code repositories (collaborative platforms).
- Collaborative workflows provide a way to organize a team around a software project.
- **Workflows: branching (centralized) and forking (shared)**
- High quality software requires good planning and management.
- Adopting concrete roles and responsibilities can help teams to organize their work.