Version Control Systems: GIT

An Introduction to Source Code Management

University of Fribourg
Department of Informatics
Software Engineering Group

October 1, 2020

Outline

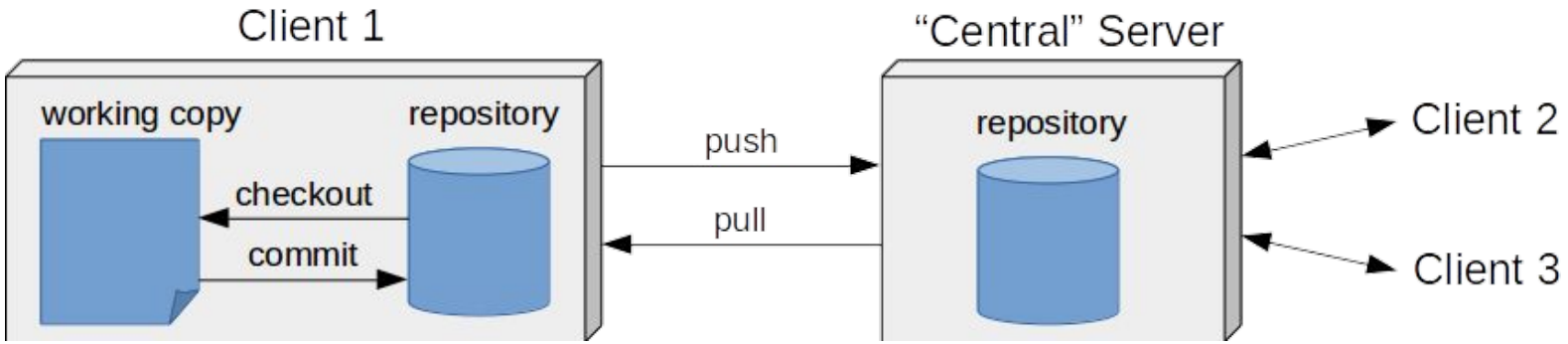
1. Introduction
2. Git Basics
3. More Git
4. Project

Version Control

- Version control is the management of multiple revisions of the same unit of information.
- Two main approaches exist:
  - The *centralized revision control*, where all the revision control functions are performed on a shared server.
  - The *distributed revision control* (DRCS), where each developer works directly with his own local repository, and changes are shared between repositories as a separate step.

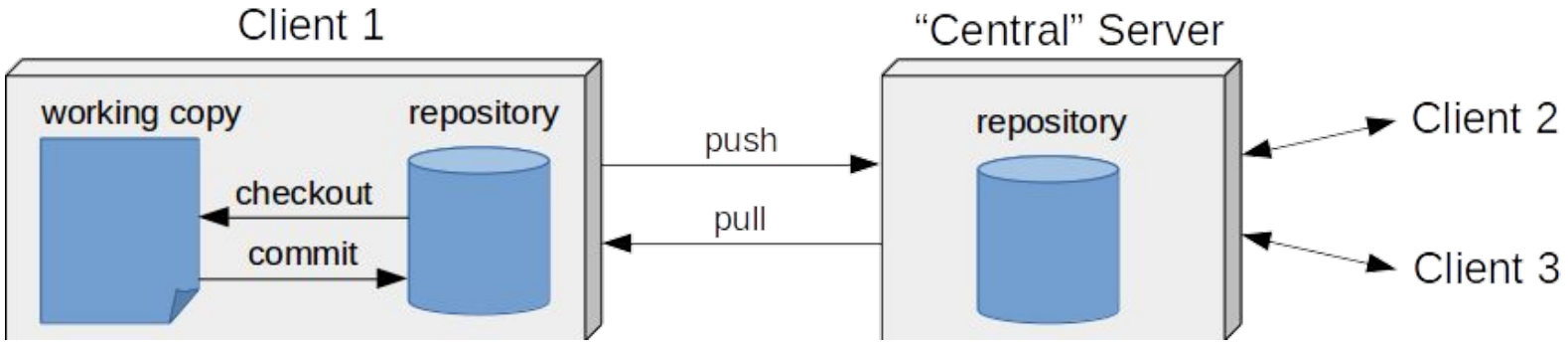
Distributed Versioning Systems I

- Each user has its own repository.
- From this repository, a user can “check out” a working copy for development, and “commit” changes back to the repository.
- Since each developer has its own repository, no mechanism is needed to prevent overwriting others’ work.
- The repository records the history of all source files. This is useful to retrace the history of a component or to bring back a trashed feature.



Distributed Versioning Systems II

- The different repositories are synced through the mechanisms of “pushing” changes to others and “pulling” others’ changes into the own repository.
- To prevent overwriting others’ work, git uses a sophisticated file merge algorithm and tries to merge the different versions together.
- Normally, there exists a central repository (which is just a repository without a working copy) to which the developers push and from which they pull the code.



Git

- The Git project was initiated by Linus Torvalds to maintain the source code of the Linux kernel.
- Upon committing, git saves the whole file system and not only individual files.
- In general, a server is chosen as the “central” repository, e.g. github.com

Getting Started I

Pre-setup

- The general structure of a Git command is:
  `git [options] <command> [args].`
- You can see the general commands using:

```
1 $ git --help
```

- Before you start, you can configure git:

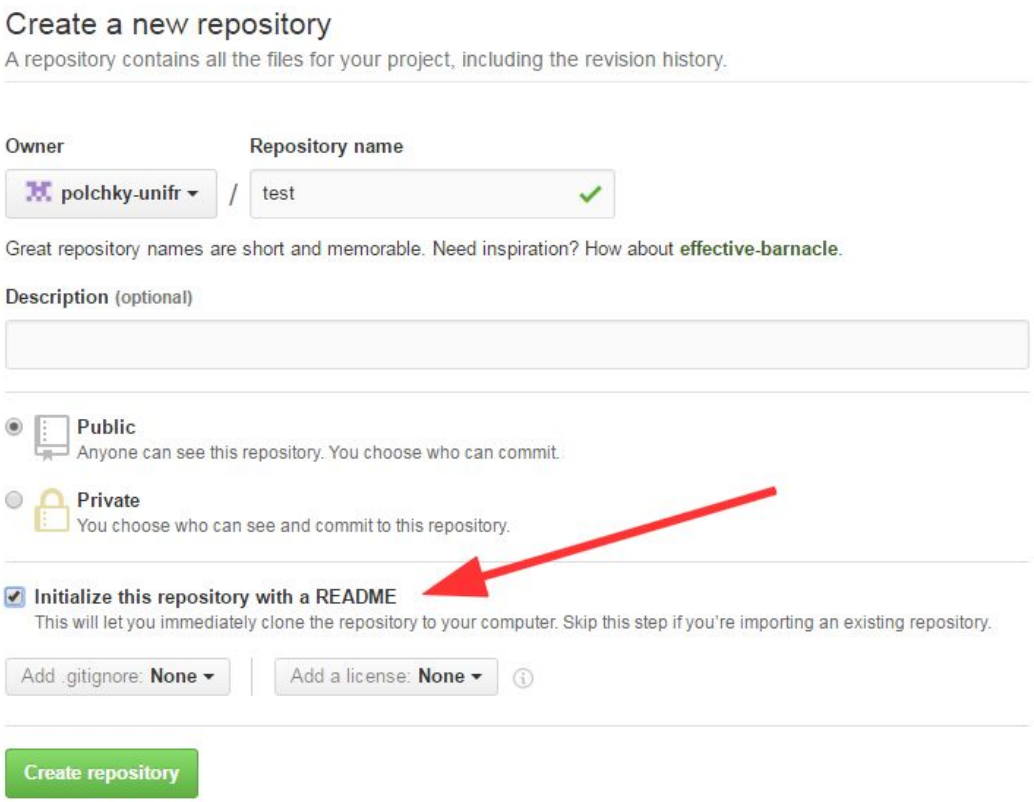
```
1 $ git config --global user.name "John Doe"
2 $ git config --global user.email johndoe@example.com
```

- For our project, Github will act as the “central” repository.

Getting Started II

Creating or joining a project on GitHub

- You can create new projects directly on github.com
- Adding a README file allows you to clone the repository on your machine.

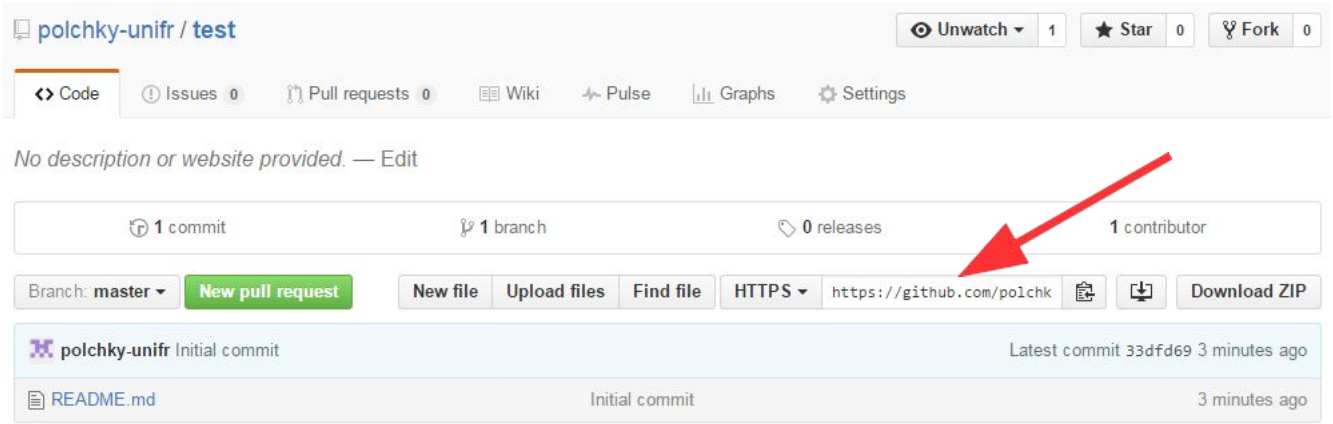


Getting Started III

Cloning a project

- To join an existing project, you need to clone it.

```
1 polchky@ubuntu:~$ git clone https://github.com/polchky-unifr/test.git
2 Cloning into 'test'...
3 remote: Counting objects: 3, done.
4 remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
5 Unpacking objects: 100% (3/3), done.
6 Checking connectivity... done.
```



Track some files, ignore others

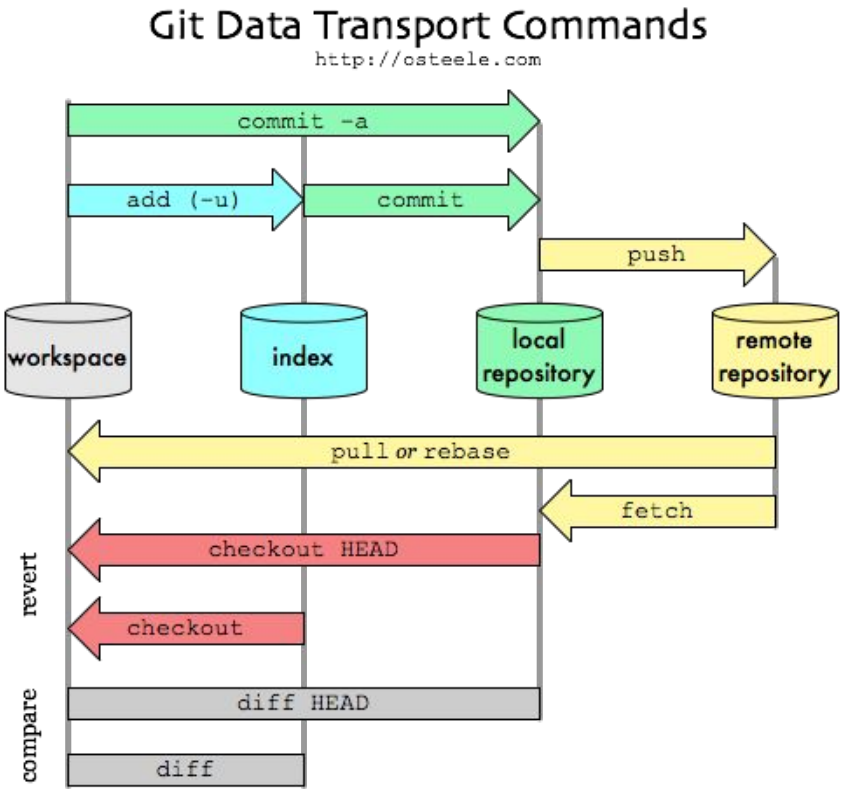
.gitignore

- Git uses a special file containing a list of strings indicating which files will be ignored by git.
- This is handy to exclude all sorts of generated files, like *.class files for a Java project.
- The project https://github.com/github/gitignore gives some default gitignore files as a starting point.
- Simply add a file named .gitignore to your repository with the files or folders you want to exclude:

```
1 bin/
2 gen/
3 nbproject/
4 reports/
5 dist/
6
7 *.class
```

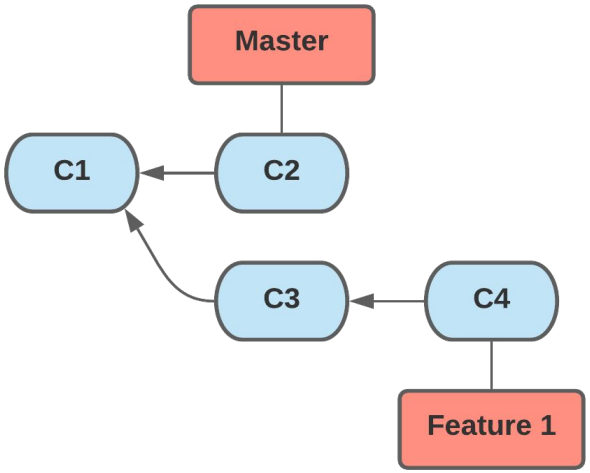
Basic Operations

- Example during the demo



Branching I

- Branches allow to better organize your code.
- Different tasks can be done in parallel using branches

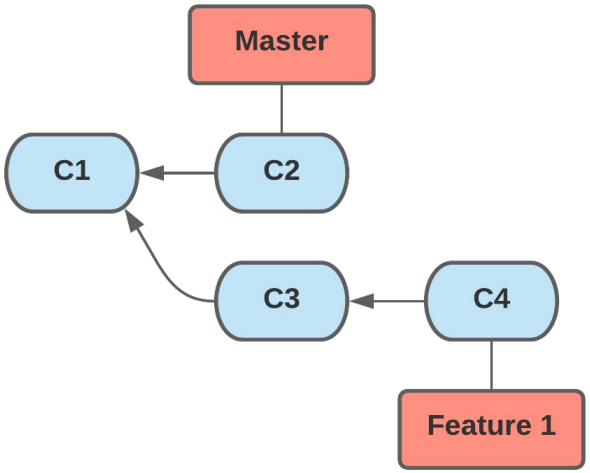


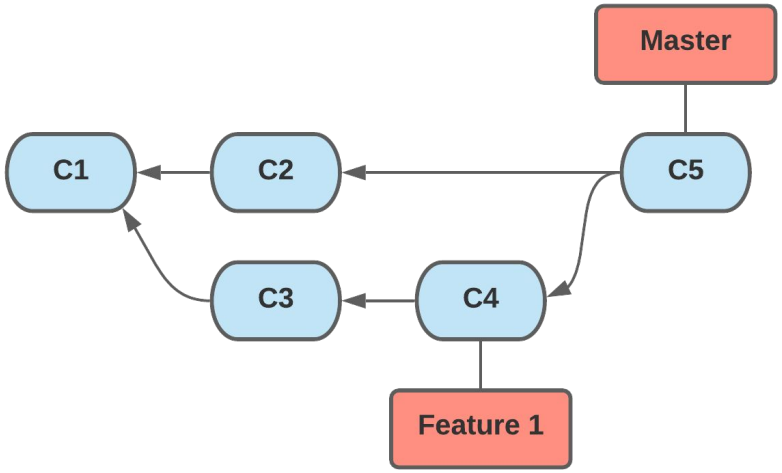

```
$ git checkout -b Feature1
Switched to a new branch 'Feature1'
```

Branching II

Merging

- In order to integrate changes made in a branch, it is necessary to merge the branch to another one.
- Some conflicts may appear and need to be solved.



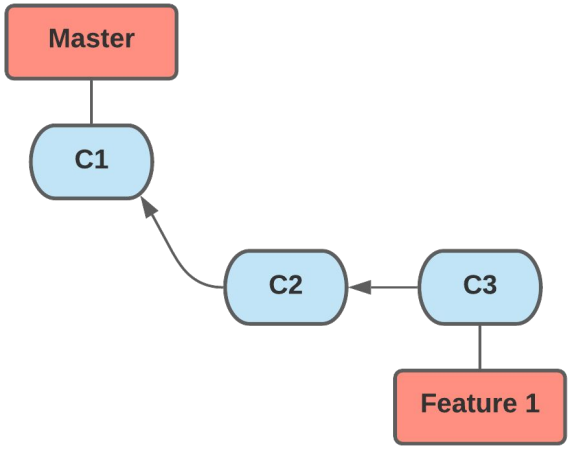


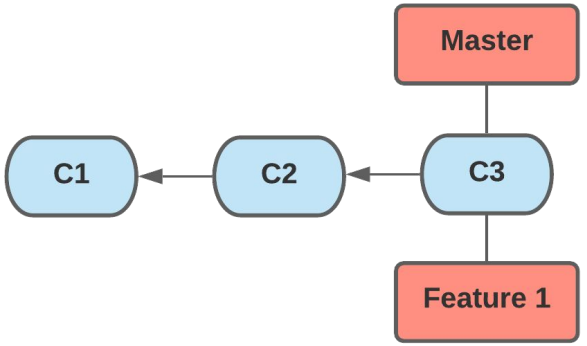

```
1 $ git checkout master
2 Switched to branch 'master'
3 $ git merge Feature1
```

Branching III

Fast Forwarding

- By default, Git fast-forwards a merge operation if no commit is present on the merged branch only.
- While practical, this behavior rewrites your branches history!



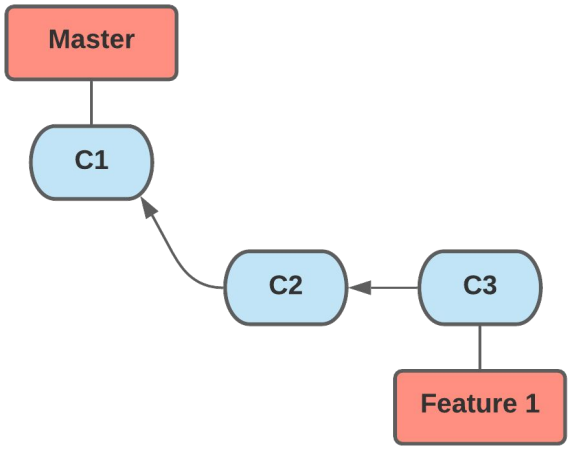


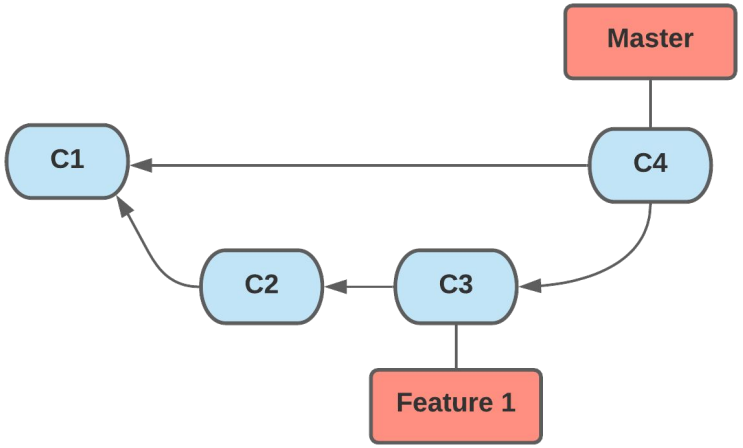

```
1 $ git checkout master
2 Switched to branch 'master'
3 $ git merge Feature1
```

Branching IV

Avoiding Fast Forward

- Avoid fast-forwarding by using the “--no-ff” option.
- You can configure git to do this automatically:
  $ git config --global --add merge.ff false



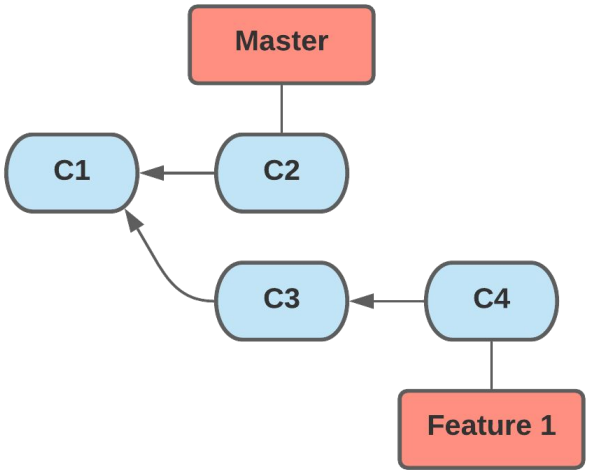


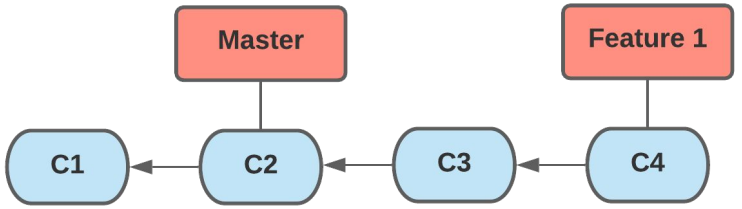

```
1 $ git checkout master
2 Switched to branch 'master'
3 $ git merge --no-ff Feature1
```

Branching V

Rebasing

- Rebasing is another way to integrate changes made in a branch by “flattening” it.
- This operation also rewrites your commits history.






```
1 $ git checkout Feature1
2 Switched to branch ‘Feature1'
3 $ git rebase master
```

Branches IV

Best Practices

- The main question when using a version control system is how to organize it. When can/should branches be created and how many of them?
- A good practice is to have one main branch (the trunk) which contains all the release versions (e.g. the stable ones) and a development branch. When development of a feature becomes stable, one can merge it into the trunk.
- Please read this excellent article about git branching management: https://nvie.com/posts/a-successful-git-branching-model/

Thingy Project I

- For your project, we ask you to create repositories on github (one per entity of your system):
  - thingy-api-<group name>
  - thingy-client-<group name>
  - thingy-workflow-<group name>
  - ...
- (The group names will be given by us next week)
- Add us as collaborators on projects:
  - arnaud.durand@unifr.ch
  - pascal.gremaud@unifr.ch

Thingy Project II

- We will use github as one of the tools to judge the quality of your projects:
  - Frequency of commits
  - Structure of branches
  - Repartition of commits by members
  - …
- At the end of each sprint, we will ask you to produce a release with tag “v1”, “v2” and “v3” respectively, **for your api only.**
  - We will run your api, but we only need a demo of your clients, during class.