ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

01
Chapiter

Version Control with Git

2I1AC3 : Génie logiciel et Patrons de conception

Régis Clouard, ENSICAEN - GREYC

“Git signifie ne jamais avoir à dire que vous êtes désolé.”
M. J. Dominus

Outline

1

Why

Version Control?

Why Version Control?

1) **For tracking and managing changes to files**
   - Change log : what is new (addition, modification, deletion of data) from previous version
   - Rollback to previous version (undo / redo)

Why Version Control?

1) For tracking and managing changes to files
2) **For managing several versions at the same time**
    - eg. Microsoft OS: Windows 11, Windows 10 Home, Windows 10 Pro, ...

Why Version Control?

1) For tracking and managing changes to files
2) For managing several versions at the same time
3) **For sharing code**
   - Integrate developer's contributions to build a common version of the software
   - Identify and resolve conflicts when two developers modify the same file.

Why Version Control?

1) For tracking and managing changes to files
2) For managing several versions at the same time
3) For sharing code
4) **For organizing collaborative work: workflow**
   - Organize how to develop software with multiple contributors
     - eg. Linux with its 1,500 contributors!

Outline

1
Why
Version Control?

2
The Git
tool

Git

- History
  - 2005 Linus Torvalds (creator of Linux)
- Alternatives
  - Mercurial
  - Perforce (video game)
  - Subversion
- Most popular
  - Free, open-source, super fast, scalable
  - Should be mentioned in your resume
- Execution
  - Command line: all commands start with word ‘git’:

```
$ git <command> [--<parameter>]* <arg>*
```

  - Interfaced by IDE and dedicated software tools

First Time with Git

- Set the git variables:

```
$ git config --global user.name "Blaise Pascal"
$ git config --global user.email Blaise.Pascal@ecole.ensicaen.fr
$ git config --global core.editor vim
```

Outline

1
Why
Version Control?

2
The Git
tool

3
Local Repository

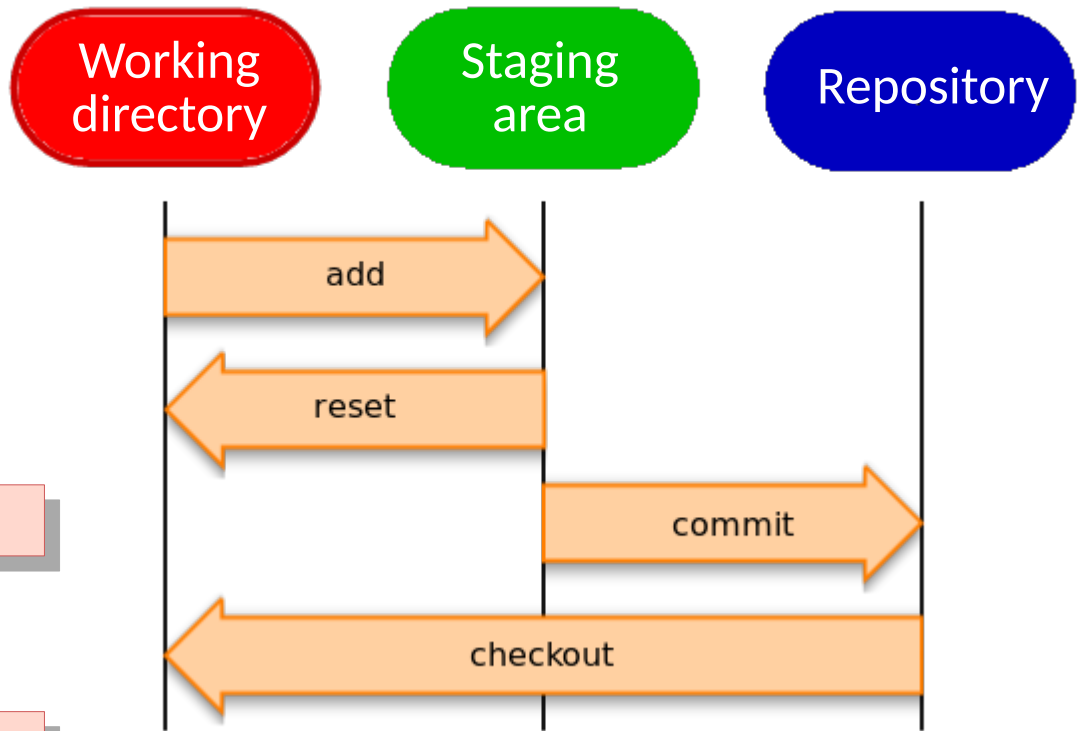
Create Local Repository

- Create repository

```
$ mkdir <project>
$ cd <project>
$ git init
```

Local Repository

- File lifecycle: 3 sections
  1) Working directory
  2) Staging area (aka index)
  3) Repository



- **Hint**: get the state of each section

```
$ git status
```

  - Short version

```
$ git status -s
```

Working Area → Staging Area

- Add files in the staging area
  - Note: the files in the staging area are fresh copies of those in from the working area

```
$ git add <filename>
$ git add -A # stages all changes (creations, modifications, deletions)
             # in the entire project directory
$ git add .  # stages all changes (creations, modifications, deletions)
             # from the current directory and subdirectories
```

- Remove files from the staging area

```
$ git reset <filename>
```

- Show the contents of the staging area

```
$ git ls-files
```

Working Area → Staging Area

- File `.gitignore` (operates on its directory and subdirectories)
  - Specifies the files and folders that you do not want to commit in the repository
  - The "git add" command becomes inoperative on the specified files

```
# example of a .gitignore file
local.properties      # a single file
*.bak                 # many files
etc/                  # folder
app/build/            # folder
```

Staging Area → Repository

- Move the files from the staging area to the local repository

```
$ git commit -m "<Message justifying the commit>"

For example:
$ git commit -m "Added User table in the database"
```

  - Without the *-m* argument, git opens the editor
  - In one command: git add + git commit

```
$ git commit -am "<Message justifying the commit>"
```

    - **Caution**: do not take into account new files

- Message relevance is important because it is the only clue to understand the history and therefore to find into it.

Staging Area → Repository

- Remove the file from the working directory and only the Git repository.

```
$ git rm <file>
```

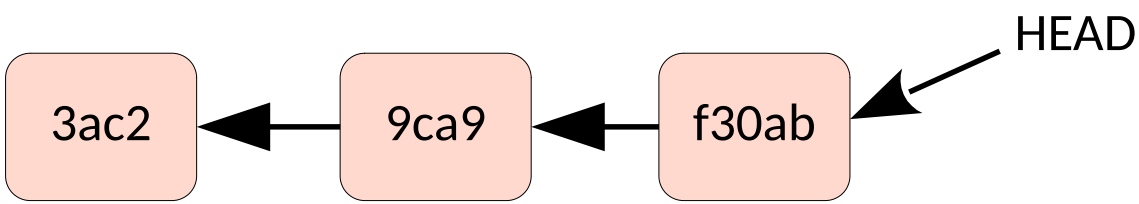
- Remove the file only from the Git repository, but not from the working directory.

```
$ git rm --cached <file>
```

- **Hint**: Use –-*cached* for removing files in the repository after added them in the *.gitignore*

Commit

- Repository is organized as a linked list of commits
- HEAD points to the current commit in the repository
- Commit:
  - Records a fresh copy of all files that appear in the staging area
  - Archives and compress files in file
  - Identifies the file by a hashed number: sha-1
  - Moves the HEAD to this new file
- The list allows you to go back and forth in the linked list



History of commits

- Importance of the commit messages
  - Log:

```
$ git log
commit 1fdbb470a9ac440eb1015227fb912f40163d4688
Author: Régis Clouard <regis.clouard@ensicaen.fr>
Date:   Tue Jul 8 14:20:41 2025 +0200

    refactor(control): extract CsvGenerator from Controller

commit dca4c78cd83d883c1aa1a727c15f395a9a25a85d
Author: Régis Clouard <regis.clouard@ensicaen.fr>
Date:   Tue Jul 8 10:39:49 2025 +0200

    feat(global setting): prevent using distribution directory as working directory
```

  - Rollback to old version

```
$ git log
$ git checkout <sha-1>
```

```
$ git checkout dca4c78cd83d883c1aa1a727c15f395a9a25a85d
```

Outline

1
Why
Version Control?

2
The Git
tool

3
Local Repository

4
Demo Git
Basics Commands

Outline

1
Why
Version Control?

2
The Git
tool

3
Local Repository

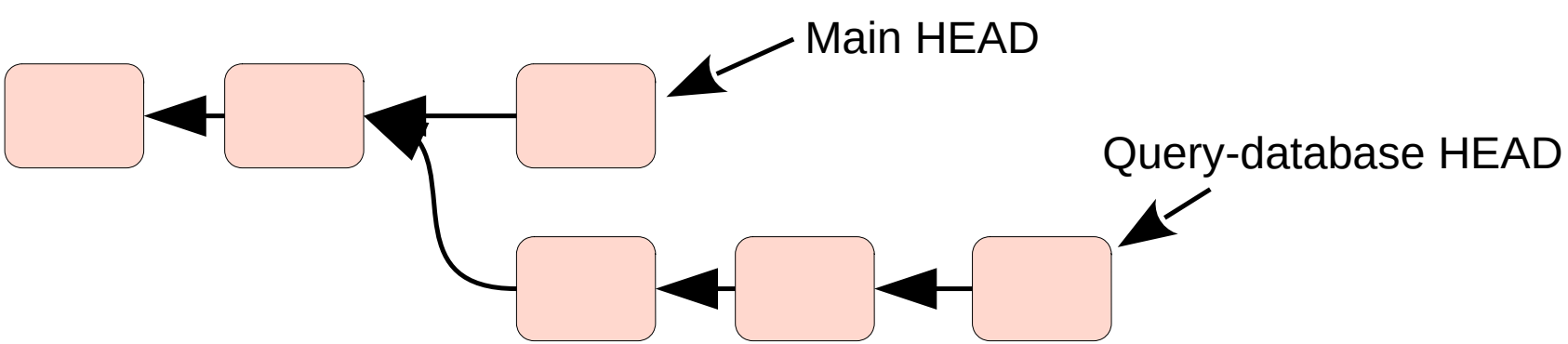
4
Demo Git
Basics Commands

5
Branches

Branch

- A branch holds a version of the repository
  - We can reference several versions in the same repository like so many branches
- By default, there is a branch named "*main*" or "*master*"
- Git command:

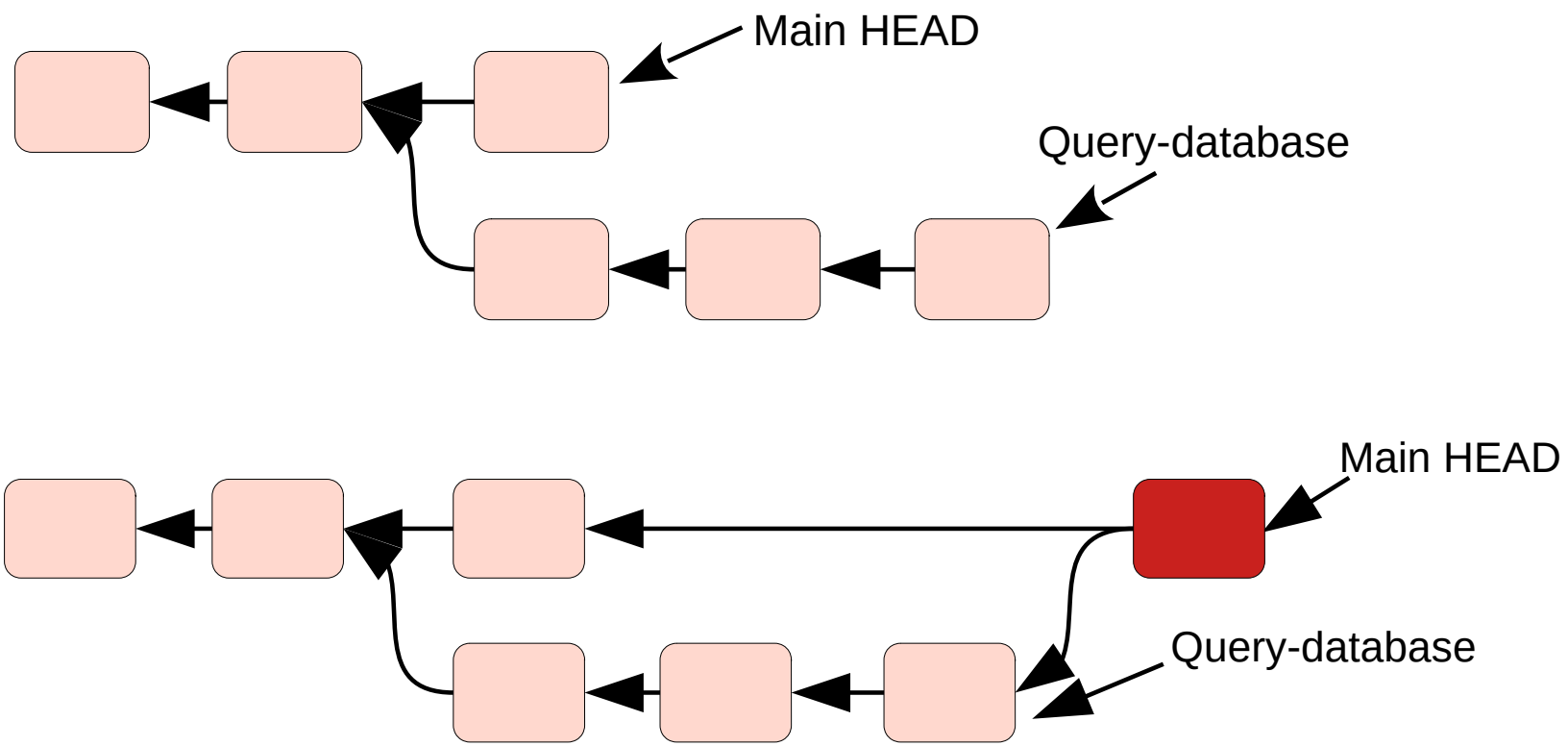
```
$ git branch query-database    # Create a branch as a copy the current branch
$ git checkout query-database  # Install a branch on the working directory
$ git checkout main
$ git branch -d query-database # Remove the branch
```



Merging branch

- Merge one branch into another:

```
$ git checkout main
$ git merge query-database
```



Conflicts

- A conflict appears when two branches are merged and some common files has been modified in both branches
- Conflict resolution must be done by hand!

Conflict Resolution

- Edit the conflicting file and delete the conflict markers <<<<<<<, =======, >>>>>>> and make the changes you want in the final merge.

Before

```
#include <stdio.h>
#include <stlib.h>
<<<<<<< OURS
void do_something_else() {
=======
void do_something() {
>>>>>>> THEIRS
    // code non modifié
}
```

Version of the current contributor named OURS

Version of the merge request contributor named THEIRS

After

```
#include <stdio.h>
#include <stlib.h>
void do_something() {
    // code non modifié
}
```

New version without conflict after keeping the "theirs" version

Outline

1
Why
Version Control?

2
The Git
tool

3
Local Repository

4
Demo Git
Basics Commands

5
Branches

6
Demo
Branches