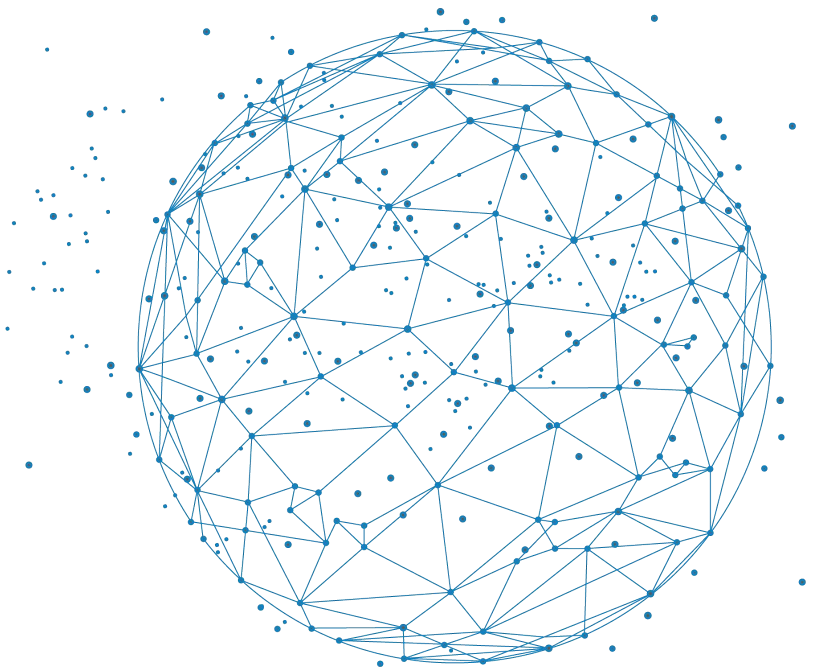


Initiation à Git

J. Seiler

Plan

- What is Git ?
- Key concepts
- How to use the Git command ?
- The Burger project
    - Create a repository
    - Add files
    - Commit changes
    - Modify files
    - Check status
    - Revert changes
    - Use branches

This is **an interactive training**

- You will learn to use `git` by working with it
- You only need a web browser and an Internet connection to follow this training.
- Each time you see this icon, you will have something to do

What is git ?

git is

a command line tool	*command line(git)* *developed in C, Bash and Perl* *Open Source (GNU GPL 2)*
a version control system	- track changes to a file/folder or a set of files/folder - navigation in the history of modifications - sharing of changes
decentralized	no need for a server
and distributed	multi-user

Where does git come from ?

Git was invented in 2005 by Linus Torvalds.

Git is the successor of many similar tools like cvs or subversion.

git means « unpleasant person »
(Linus like to name his projects after himself…)

A few examples

- Follow the steps of modification of a program
- Test a complex change and be able to go back easily
- Working with others on a project
- Invite collaborators on a project
- Contribute to OpenSource projects

How to use git ?

- git is **a command line tool**
- the command is **git**
- you can use it from shell terminal

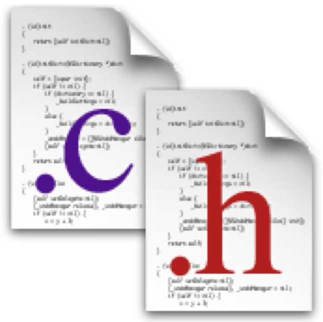
How does git work ?

For each project you track with git, git maintains a repository at the root of the project in a `.git` folder

The `git` command let you interact with this repository.

Working directory **vs.** Repository

Working directory



Your working folder contains the files and folders that make up your project.

Git can modify these files to update them or present them to you at different versions of the project through its index.

Git repository

.git

Your git repository contains the entire history of your project. All file versions, all modifications, etc.

This is the `.git` folder at the root of your working folder.

Okay… but **what is an index ?**

A Git repository will allow you to track the **history of changes** in your project.

Each change is first recorded in an index (or indexed) to form **a collection of changes**.

This collection of changes is then validated (or committed) in your repository

Each commit is a new version or revision in your project history.

Before we start…

To use the latest version of git, load the command using `module`

```
$ module load git
$ git --version
git version 2.40.1
```

How to use git ?

In your console, type `git`

```
$ git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone             Clone a repository into a new directory
   init              Create an empty Git repository or reinitialize an existing one



work on the current change (see also: git help everyday)
   add               Add file contents to the index
   mv                Move or rename a file, a directory, or a symlink
   restore           Restore working tree files
   rm                Remove files from the working tree and from the index
   sparse-checkout   Initialize and modify the sparse-checkout

[…]

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.
```

How to use git ?

The `git` program allows you to run commands to manage your git repository

```
$ git <commande> <arguments>
```

Defining your identity

Your identity will be associated with the changes you make in your repositories

It is defined in the file `~/.gitconfig` or `%USERPROFILE%\.gitconfig`

```
$ git config --global user.name "Your name"
$ git config --global user.email your@email
$ git config --global init.defaultBranch main
```

The **burgers** project

To illustrate this training, we will work on a burgers recipes project.

Create a `burgers` folder in your home folder

```
$ mkdir burgers
```

Créer **votre dépôt**

Go to your burgers folder and run the
`git init` comand

```
$ cd burgers
$ git init
Initialized empty Git repository in /Users/seilerj/burgers/.git
$ ls -a
.     ..    .git
```

`git init <working dir>`

creates a git repository

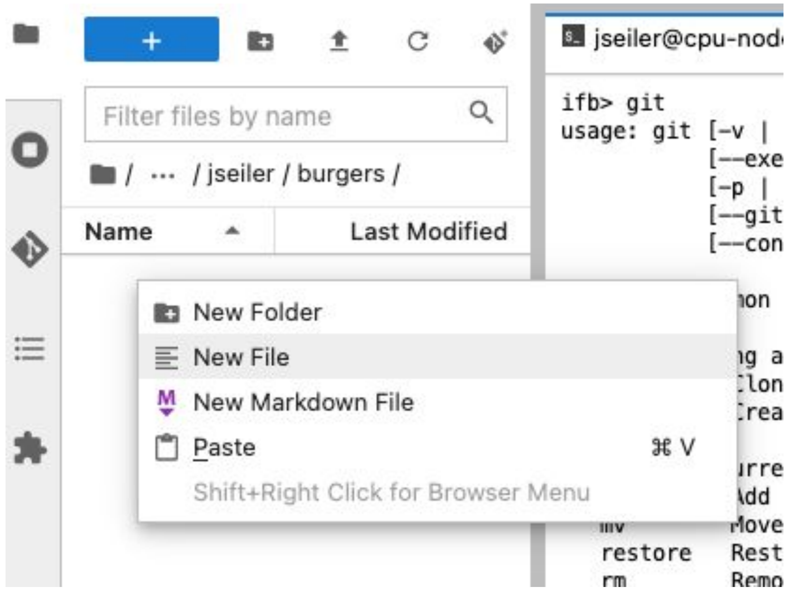
If the specified working dir does not exist, it will be created.

Without parameters, the command creates a git repository for the **current folder**.

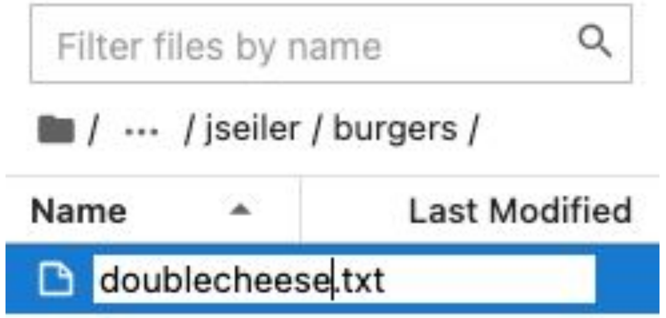
Add a file to your repository

Let’s cook a burger

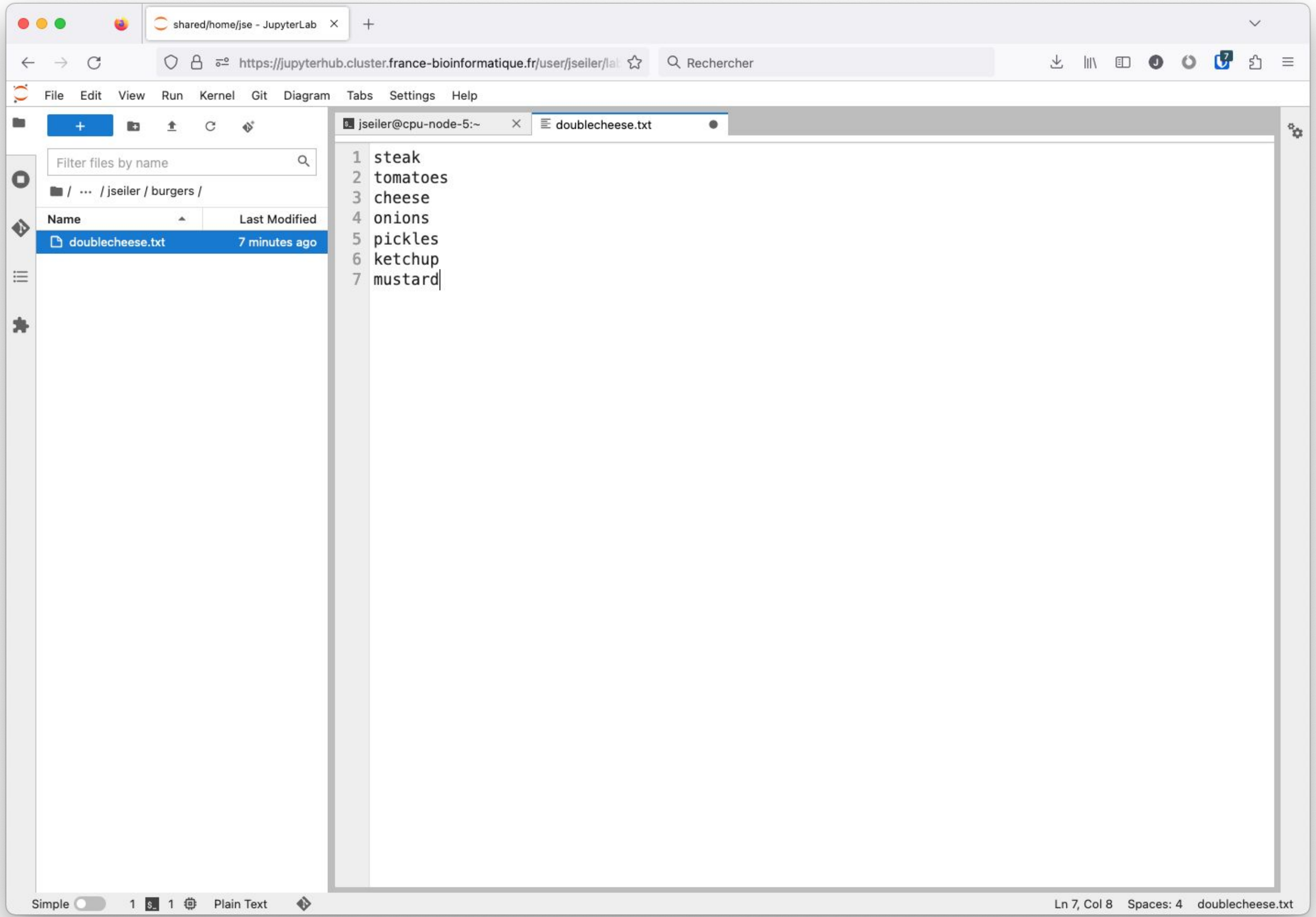
Create the file doublecheese.txt in the burgers folder and write down the list of ingredients to make a double cheese.



```
ifb> git
usage: git [-v |
           [--exe
           [-p |
           [--git
           [--con
```



Add a file to your repository



```
steak
tomatoes
cheese
onions
pickles
ketchup
mustard
```

Index a file in your git repository

Index the doublecheese.txt file in your repo with the **`git add doublecheese.txt`** command

```
$ git add doublecheese.txt
```

`git add <file(s) or folder(s)>`

adds one or more files/folders to the modification index

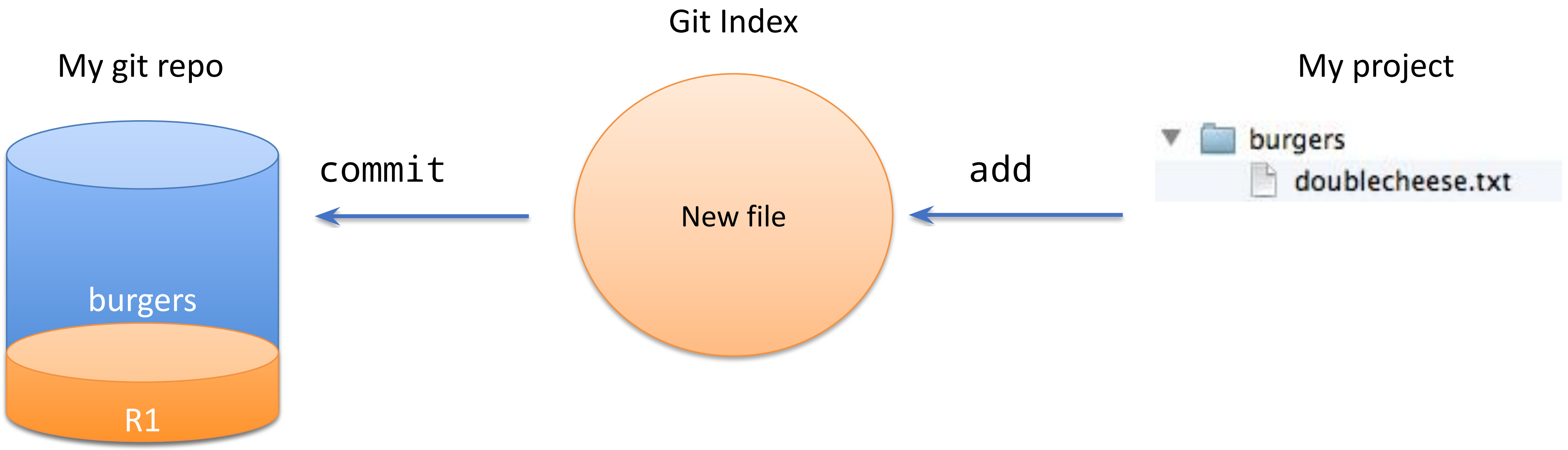
Beware: the files/folders are just marked but their content is not yet saved in the repository

Save changes

Validate this modification in order to save it in your repository with the `git commit` command

```
$ git commit -m "Birth of the double cheese"
[main (root-commit) bb0188d] Birth of the double cheese
 1 file changed, 7 insertions(+)
 create mode 100644 doublecheese.txt
```

Add a file to your repository



`git commit -m “commentaire“`

saves the changes contained in the index to your repository (by creating a new revision)

`git commit -a` saves all current modifications even if not indexed (but only for files already added to the repo)

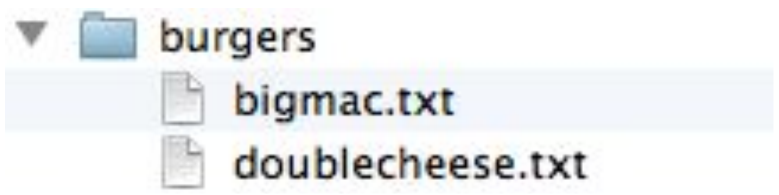
Modify a file

There are no tomatoes in the double cheese!

Correct the file `doublecheese.txt`

A new burger

Add the file bigmac.txt to your burgers project



steak
salad
tomatoes
onions
pickle
ketchup
mustard

Where are we now?

Check what has changed in your project with the **git status** command

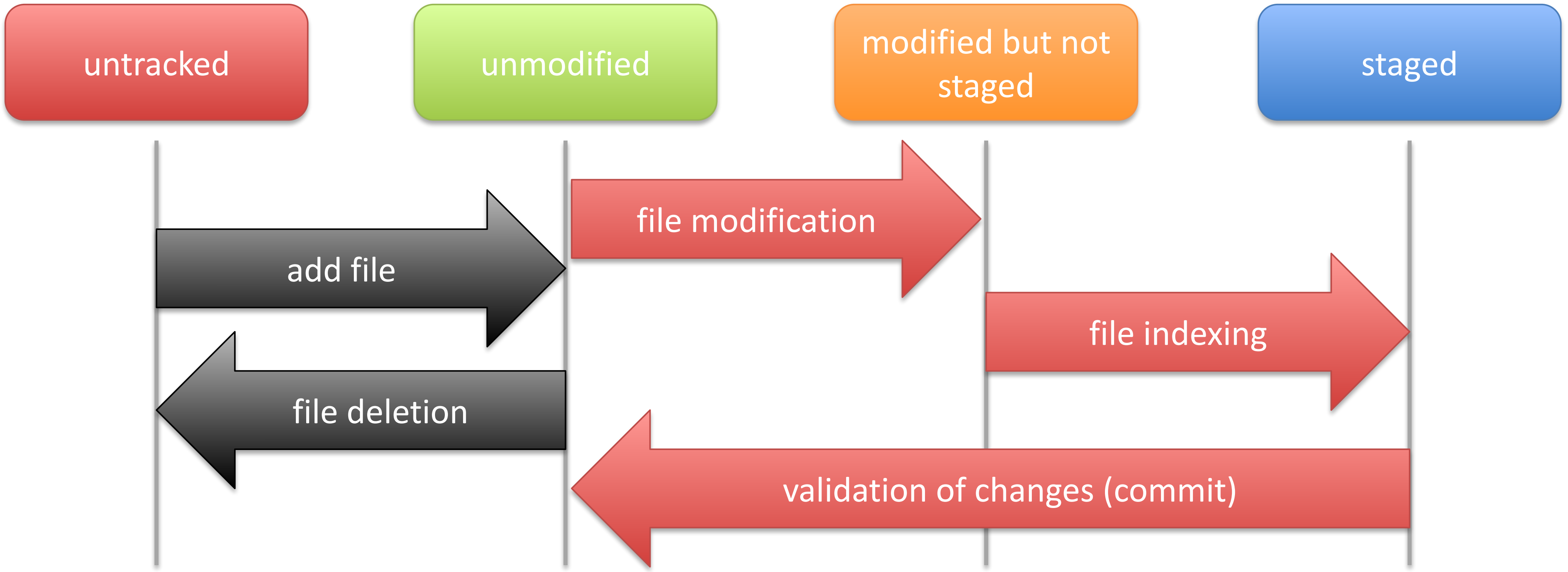
```
$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   doublecheese.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .ipynb_checkpoints/
        bigmac.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

`git status`

allows you to list the current modifications (not saved in your repository) in your working folder. There are 4 possible states for each file:

- untracked (unknown)
- unmodified
- not staged (modified but not added to the index)
- to be commited (added to the index)

Files **status** in a Git repository



Index changes

Index the change on the doublecheese.txt file with the **`git add`** command

```
$ git add doublecheese.txt
```

Check the **status of your project**

Check what has changed in your project with the `git status` command

```
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   doublecheese.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
         .ipynb_checkpoints/
        bigmac.txt
```

Add a new file to the repo

Add the new file bigmac.txt with the `git add` command

```
$ git add bigmac.txt
```

Check the **status of your project**

Check what has changed in your project with the **`git status`** command

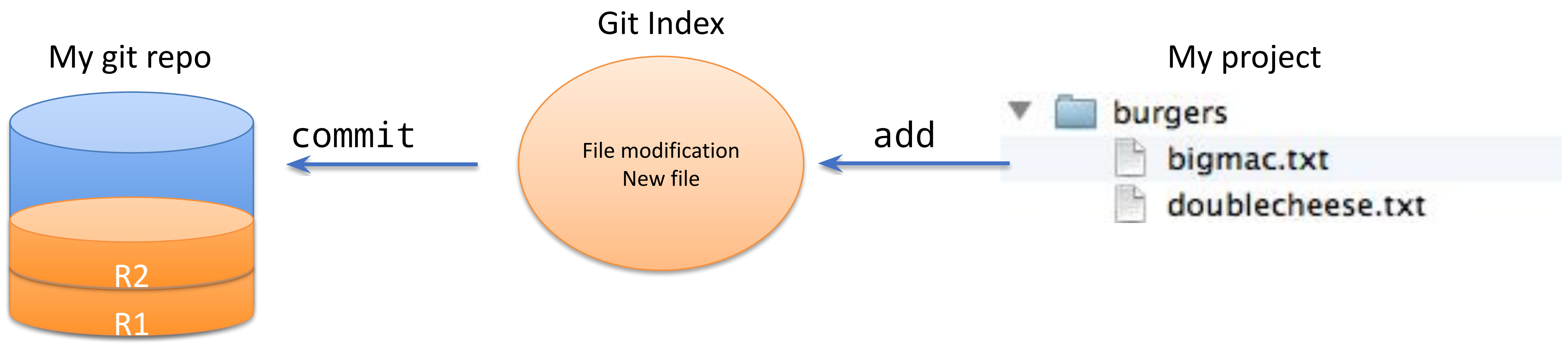
```
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   bigmac.txt
        modified:   doublecheese.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .ipynb_checkpoints/
```

Validate the changes

Run the **git commit** command

```
$ git commit -m "Add the big mac and correct the double cheese »
[main 299a6b2] Add the big mac and correct the double cheese
 2 files changed, 7 insertions(+), 1 deletion(-)
 create mode 100644 bigmac.txt
```

A new revision is here



Let’s introduce **an error**

Edit the `bigmac.txt` file to add a bad ingredient.

Retrieve **the previous version of a file**

Restore the last valid version of the file with the `git checkout` command

```
$ git checkout bigmac.txt
```

`git checkout <file or folder>`

allows you to restore a file or folder to its last version as saved in your repository.

It is possible to specify a revision number to restore the file or folder to a previous version.

Did you say revision number?

Each commit to your repository creates a new revision or version of your project.

Each revision is actually **a set of changes**.

View the revisions of your repository

Use the `git log` command to view the revisions of your local repository

```
$ git log
commit 299a6b210eed54e9f4c164b85ecbcb9ed899e6eb (HEAD -> main)
Author: Julien SEILER <seilerj@igbmc.fr>
Date:   Tue May 16 12:14:35 2023 +0200

    Add the big mac and correct the double cheese

commit bb0188df5bf0c3cb3a152e52e22df1249d52e2be
Author: Julien SEILER <seilerj@igbmc.fr>
Date:   Tue May 16 11:49:37 2023 +0200

    Birth of the double cheese
```

`git log`

allows you to view all the revisions stored in your local repository

For each revision, the following information is available:

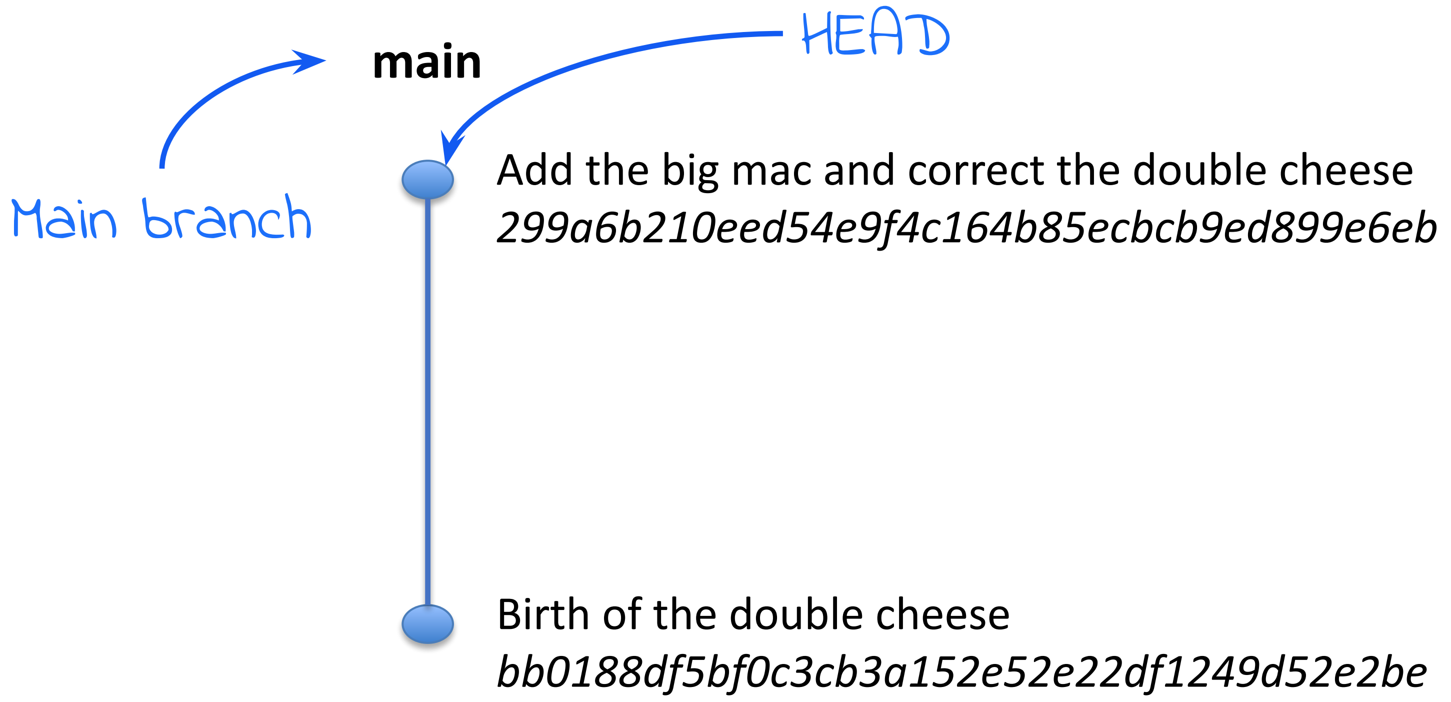
- commit : revision number
- Author : user who registered the revision
- Date : date of the creation of the revision
- Comment written by the user during the commit

Return to a previous revision of your project

Revert to the initial version of your project (just the doublecheese.txt) using the `git checkout <id rev>` command

```
$ git checkout bb0188df5bf0c3cb3a152e52e22df1249d52e2be
Note: switching to 'bb0188df5bf0c3cb3a152e52e22df1249d52e2be'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
[…]
$ ls
doublecheese.txt
```

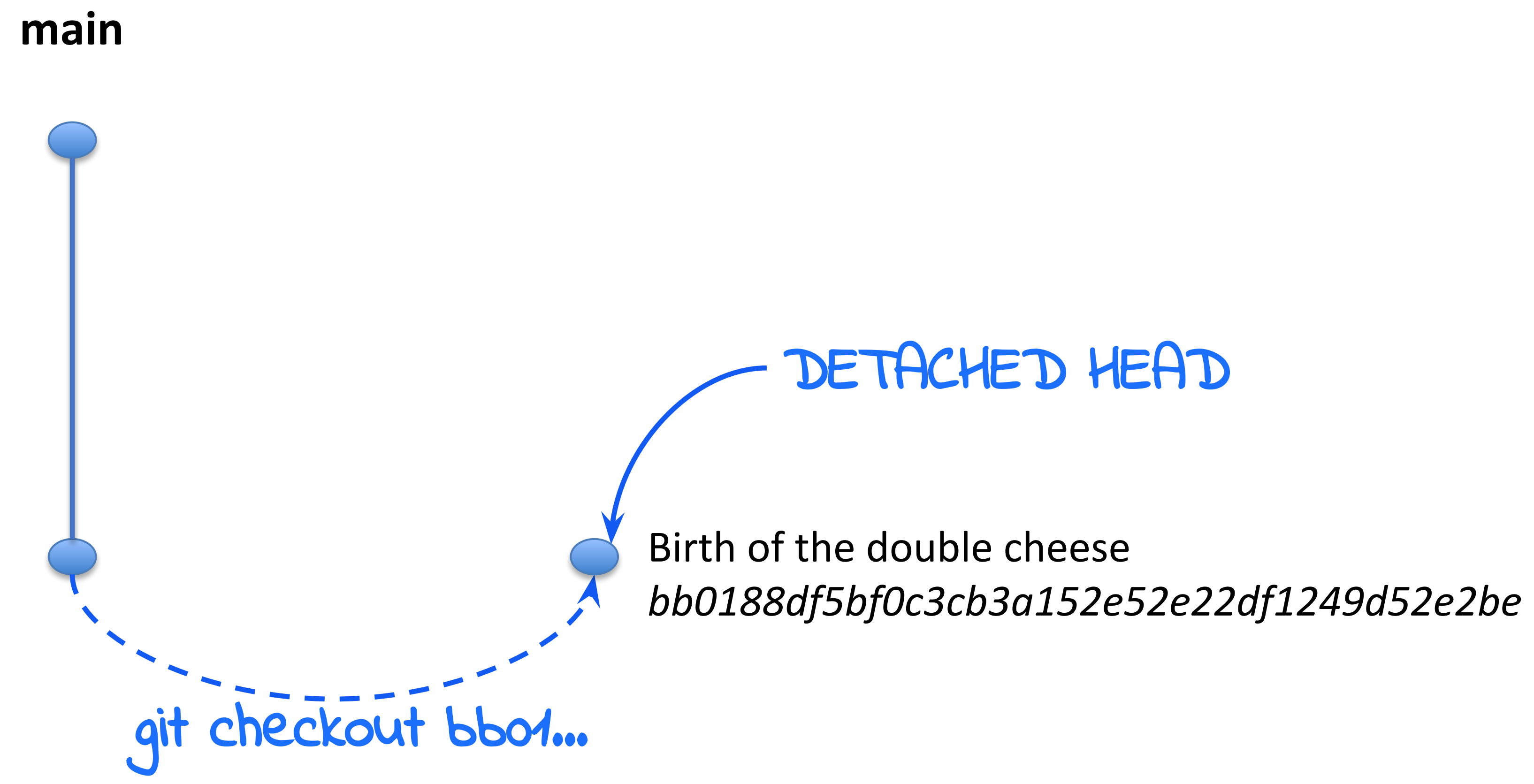
Introduction to the concept of **branch**



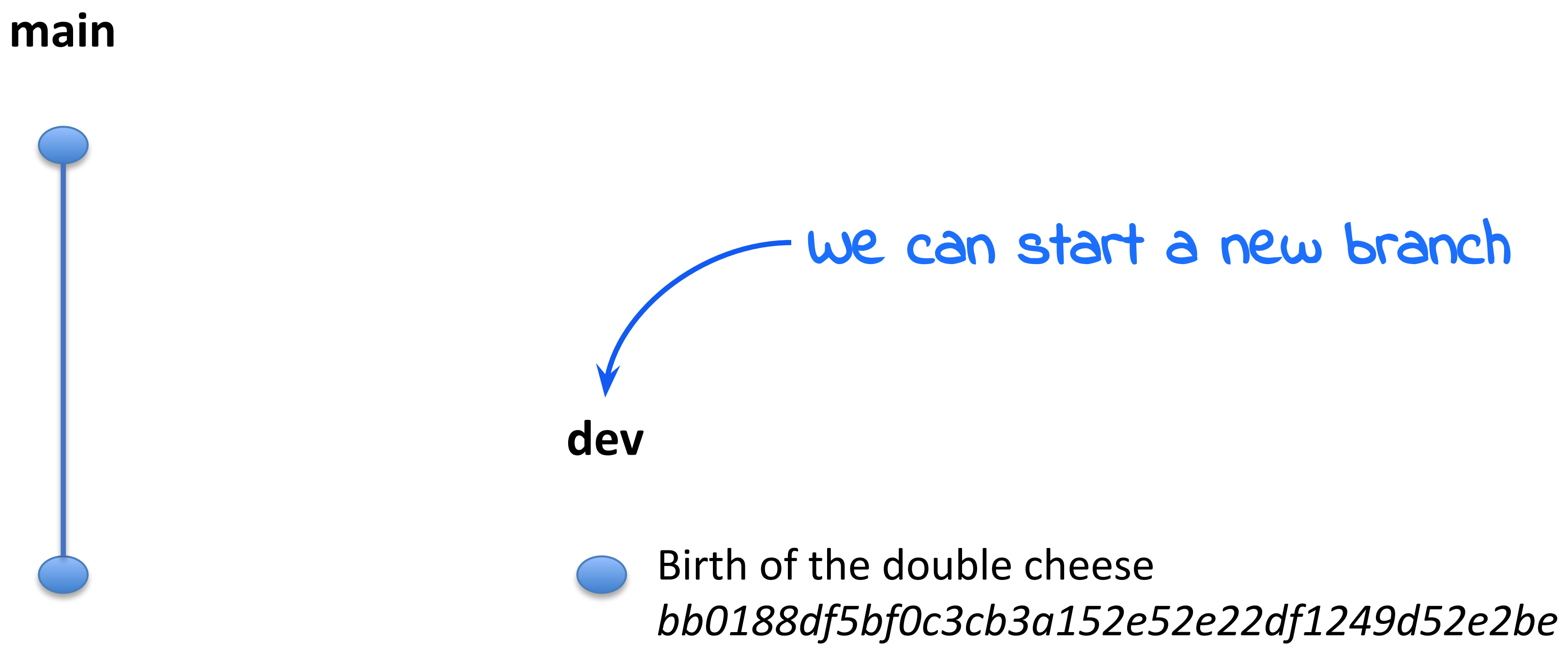
Add the big mac and correct the double cheese
299a6b210eed54e9f4c164b85ecbcb9ed899e6eb

Birth of the double cheese
bb0188df5bf0c3cb3a152e52e22df1249d52e2be

Introduction to the concept of **branch**

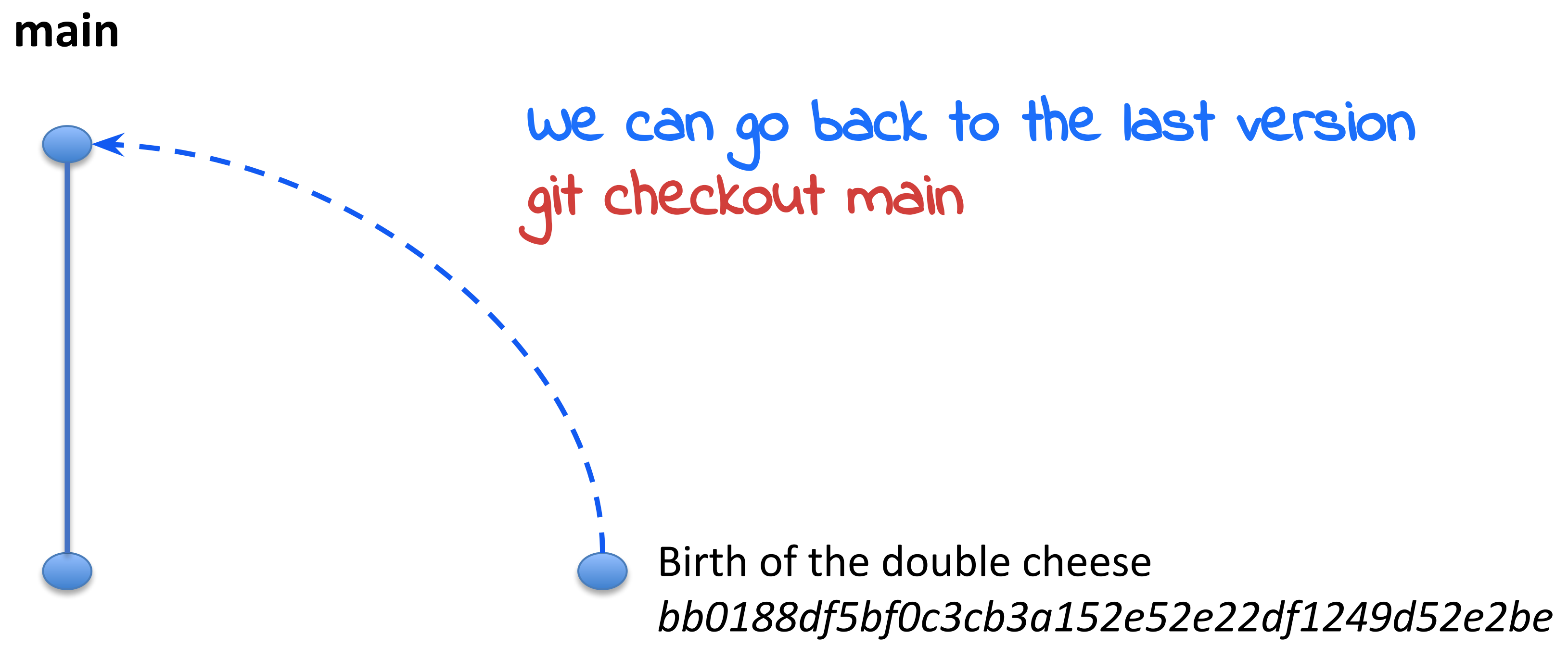

main
DETACHED HEAD
Birth of the double cheese
bb0188df5bf0c3cb3a152e52e22df1249d52e2be
git checkout bb01...

Introduction to the concept of **branch**



Birth of the double cheese
bb0188df5bf0c3cb3a152e52e22df1249d52e2be

Introduction à la notion de **branche**


main
we can go back to the last version
git checkout main
Birth of the double cheese
bb0188df5bf0c3cb3a152e52e22df1249d52e2be

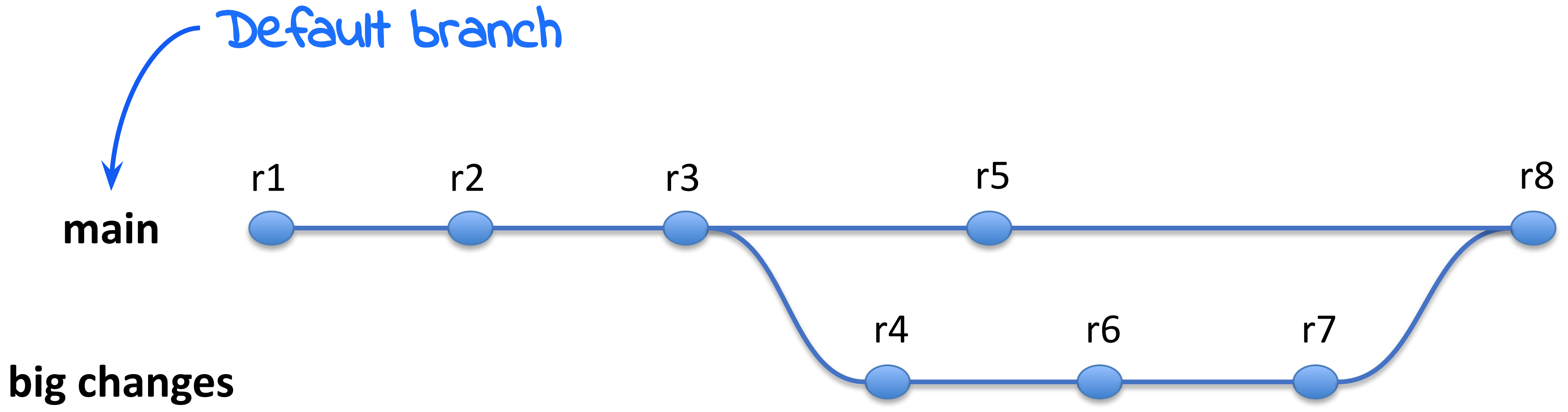
Go back to the last version of the project

Use the command `git checkout main`

```
$ git checkout main
Previous HEAD position was bb0188d Birth of the double cheese
Switched to branch 'main'
$ ls
bigmac.txt   doublecheese.txt
```

Going further with the **branches**

Going further with the **branches**



Going further with the **branches**

Our fast-food restaurant wants to go **ORGANIC**.
We need to change all our recipes!!!

That's a « big change »...

Create a new branch

Create a new `organic` branch on your repository

```
$ git branch organic
```

Create a new branch

Consult the branches available on your repository

```
$ git branch
* main
  organic
```

`git branch`

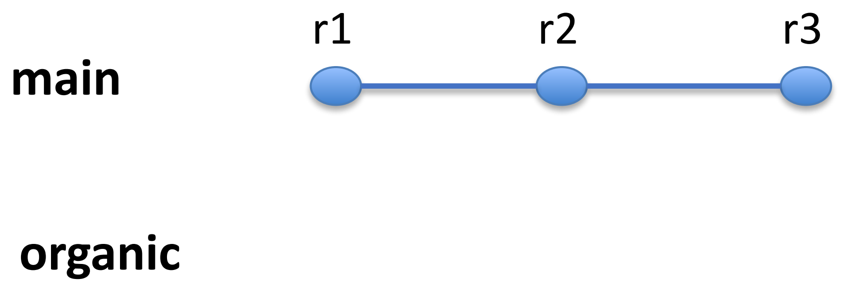
manage the branches of your repository

Change branch

Use the `git checkout` command to switch branches.

```
$ git checkout organic
Switched to branch 'organic'
```

Going further with the **branches**



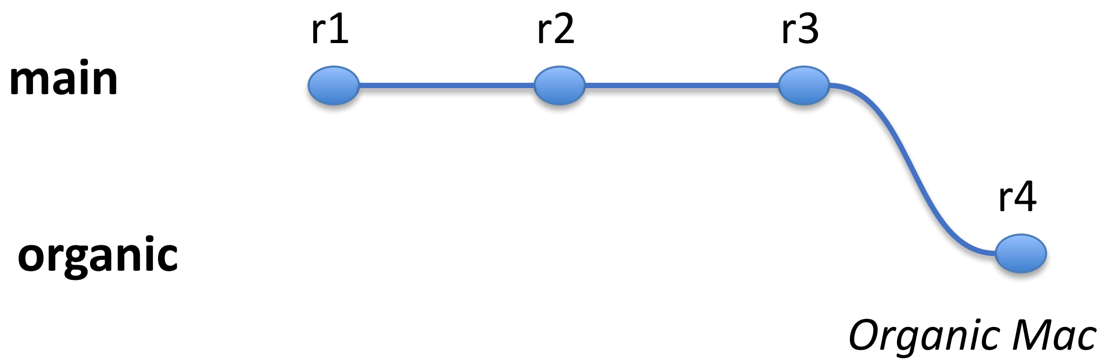
We go **organic!**

Modify the Big Mac recipe and commit the change

organic steak
organic salad
organic tomateos
organic onions
organic pickles
organic ketchup
organic mustard

```
$ git commit -a -m "organic mac"
[organic a7a6f6e] organic mac
 1 file changed, 7 insertions(+), 7 deletions(-)
```

Going further with the **branches**



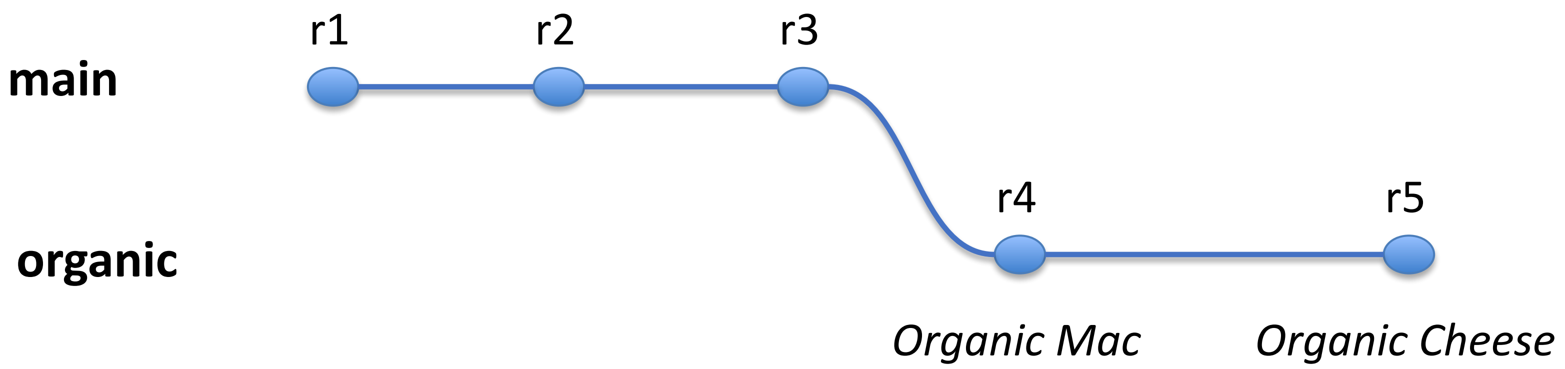
We go **organic!**

Modify the Double Cheese recipe and commit the change

organic steak
organic cheese
organic onions
organic pickles
organic ketchup
organic mustard

```
$ git commit -a -m  "organic cheese"
[organic 1ed7510] organic cheese
 1 file changed, 6 insertions(+), 6 deletions(-)
```

Going further with the **branches**



Let's go back to our **main branch**

Return to the main branch with the `git checkout` command

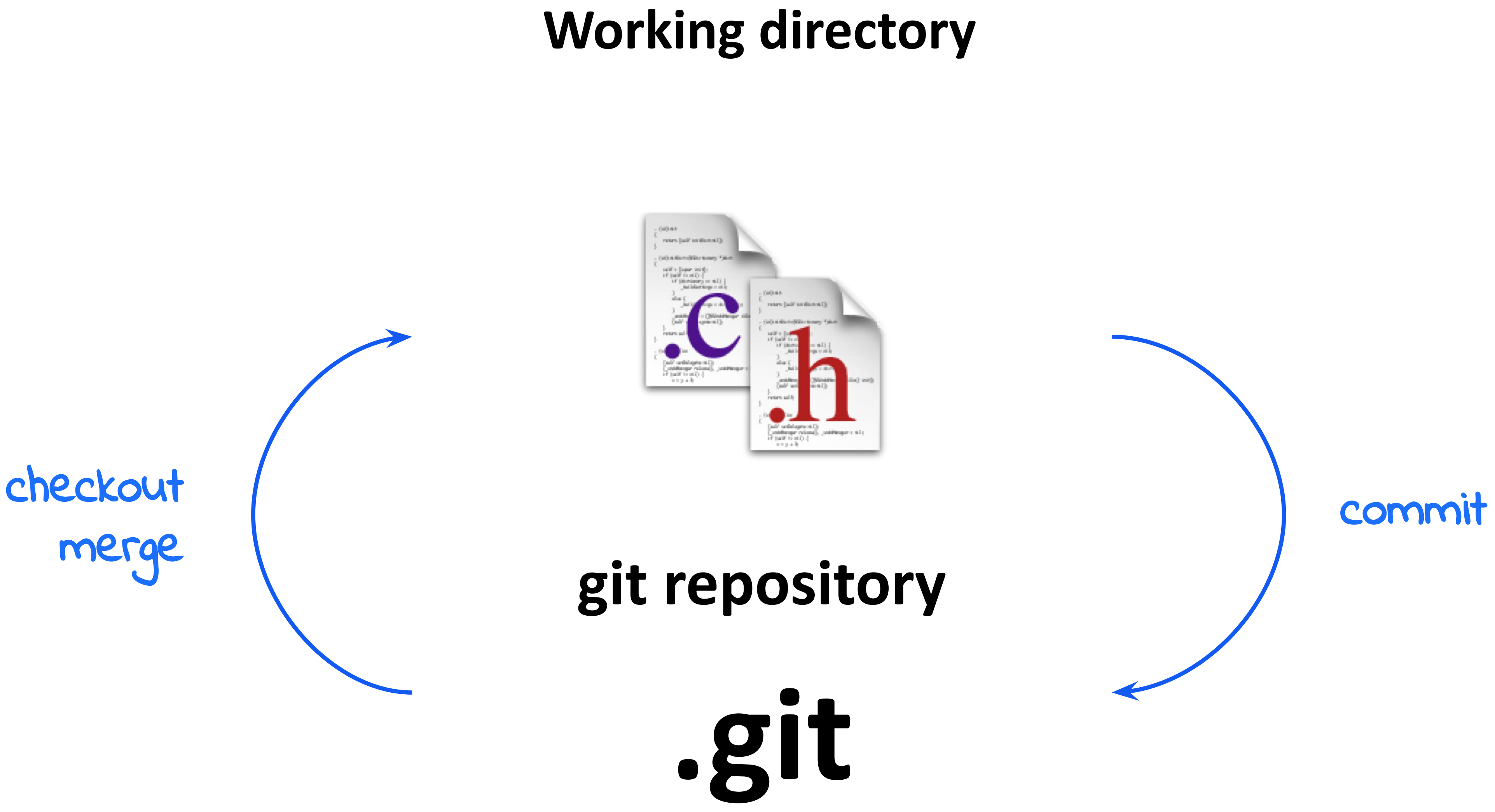
```
$ git checkout main
Switched to branch 'main'
```

Merge of branches

We can retrieve the changes saved on the organic branch with the `git merge` command

```
$ git merge organic
Updating 299a6b2..1ed7510
Fast-forward
 bigmac.txt        | 14 +++++++-------
 doublecheese.txt | 12 ++++++------
 2 files changed, 13 insertions(+), 13 deletions(-)
```

The git cycle



The git commands to manage your local repository

git command	Description
`init`	Creating a repository for a project/folder
`add`	Indexing of a modification or addition of a file or folder
`rm`	Deleting a file or folder
`mv`	Moving a file or folder
`status`	Visualization of the repository status
`diff`	Viewing changes between two revisions or between a revision and the current version
`checkout`	Retrieving a file from the repository
`log`	Consultation of the list of revisions (commits) registered on the repository

Exercice

1. Delete the bigmac and save the change in your repository
2. Add a burger and save the change to your repository
3. Restore the bigmac to your working folder

Exercice

1. Delete the bigmac and save the change in your repository

```
$ git rm bigmac.txt
$ git commit -m “bye bye big mac“
[main 916c075] bye bye big mac
 1 file changed, 7 deletions(-)
 delete mode 100644 bigmac.txt
```

Exercice

2. Add a burger and save the change to your repository

```
$ git add newburger.txt
$ git commit -m "add a new burger"
[main 26f9033] add a new burger
 1 file changed, 1 insertion(+)
 create mode 100644 newburger.txt “
```

Exercice

3. Restore the bigmac to your working folder

```
$ git checkout <rev> bigmac.txt`
Updated 1 path from 69a5a31
$ git commit -a -m "return of the big mac"
[main 9a4a1c6] return of the big mac
 1 file changed, 7 insertions(+)
 create mode 100644 bigmac.txt
```

<rev> is the last revision at which the bigmac.txt was present

Next step ? Share you work !

GitHub

GitLab