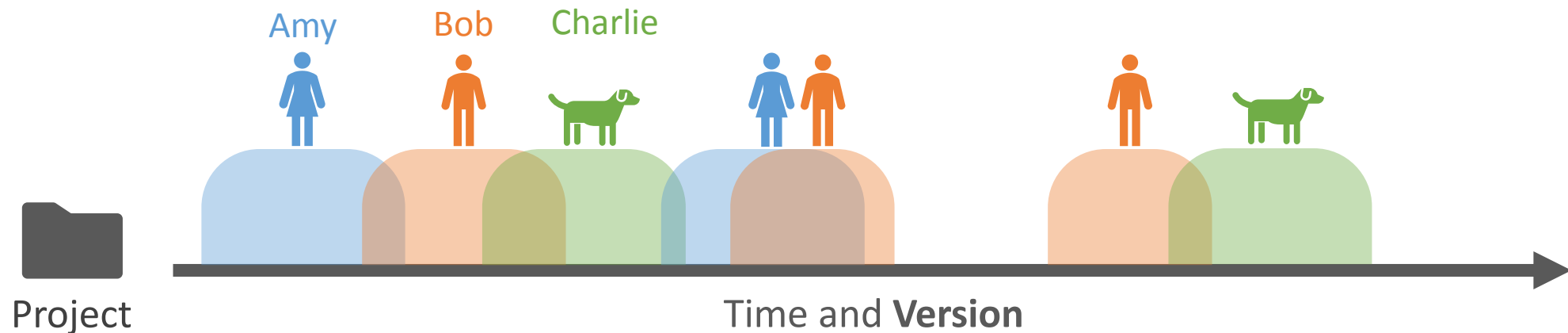# git
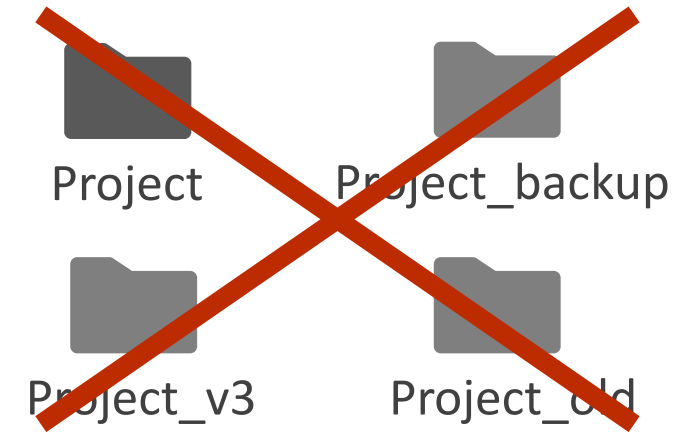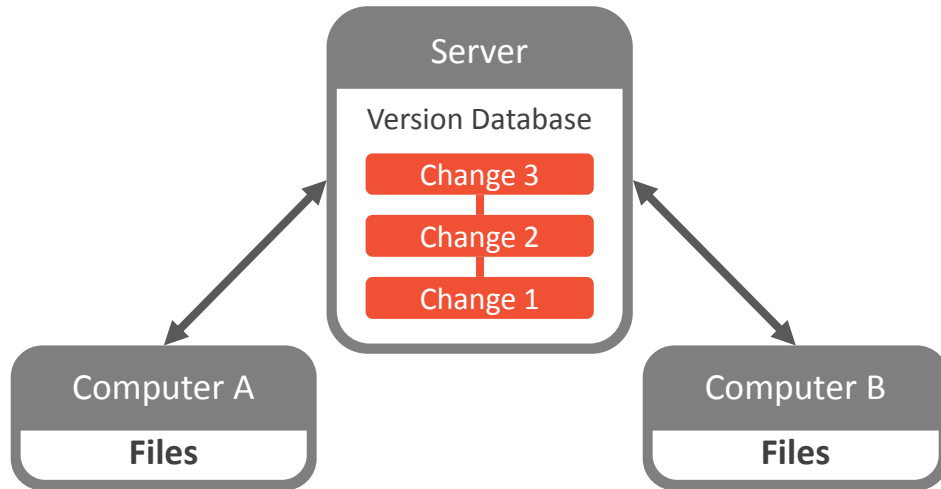
A Tutorial

Version Control

- Tracks **changes** in computer files
- Efficient **collaboration** with colleagues

Project   Project_backup

Project_v3   Project_old

Amy   Bob   Charlie

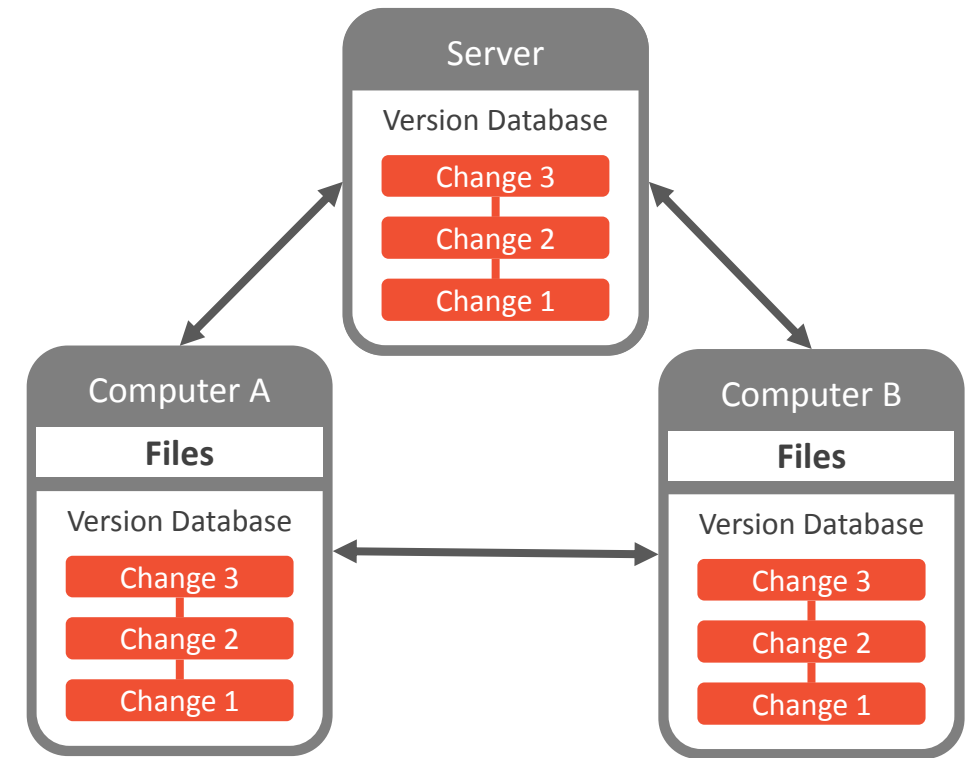Project   Time and **Version**

Version Control Systems

Centralized SVN, CVS, Perforce

+ manage access rights

Distributed Git, Mercurial
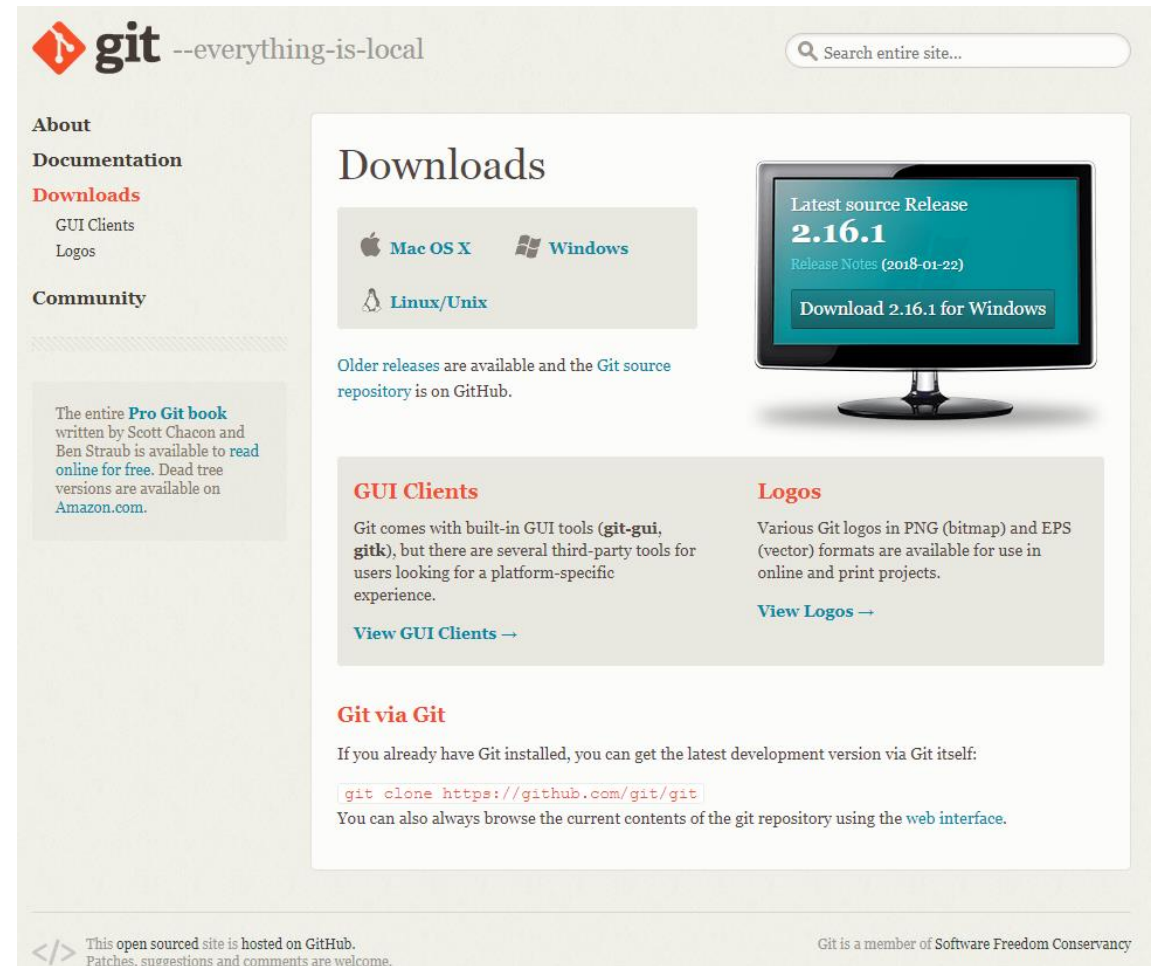
+ local version database

+ backup on collaborator machines

About Git

- Git is a **distributed** version control system for local and remote repositories

- Git is advertised with the following features
  - Offers **review** functionality
  - Work **offline** anywhere
  - **Fast** and lightweight
  - **Journal** of changes rather than a backup
  - Serves the needs of **beginners** and **advanced** users equally well

Installation

- **Download** at https://git-scm.com/downloads

Interfaces

Command Line Interface (CLI)
- Syntax: `git [verb] [options]`
- Allows for **customized** commands
- Commands are **platform-independent**

Graphical User Interface (GUI)
- Saves **typing**
- Lower **barrier-of-entry**
- **Graphically** represents collaboration

Integrated Development Environment (IDE)
- Integrates version control with **projects**
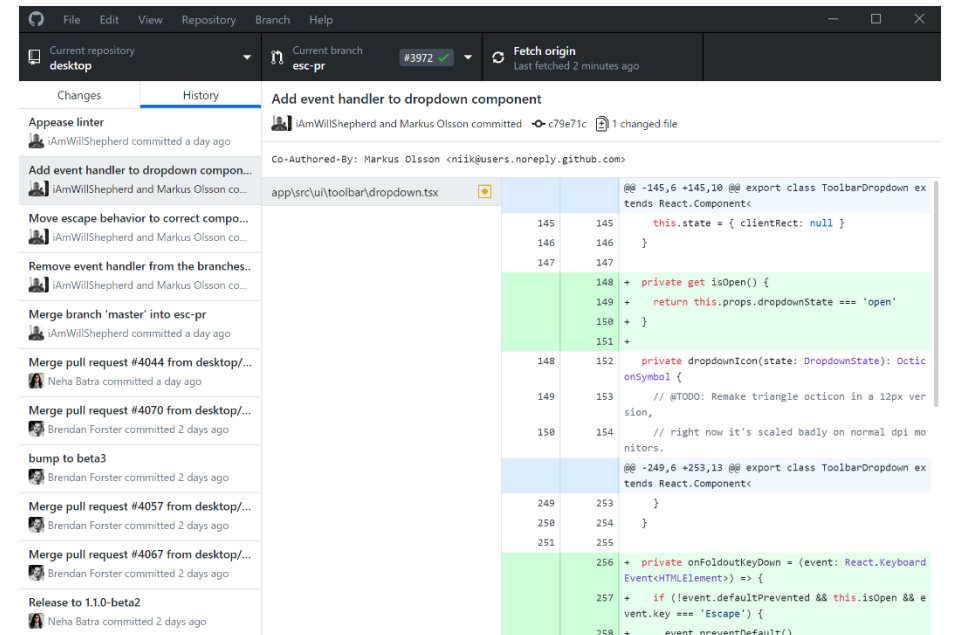
Configure Git

Verify installed git **version**

```
git --version
```

Configure git

```
git config --global user.name "your name"
git config --global user.email "my@mail.nl"
git config --list
```

Open **documentation** of verb

```
git help [verb]
git [verb] --help
```

Local Repositories

The local repository is the project folder under version control. It contains the .git subfolder and all project related files and subfolders.

Initialize local **repository**

```
git init [project-name]
```

Check s**tatus** of local repository

```
git status
```

Create empty file on Windows

```
copy NUL [file-name]
```

Create empty file on MacOs, Linux

```
touch [file-name]
```

[project-name]

.git

readme.txt

my_code.R
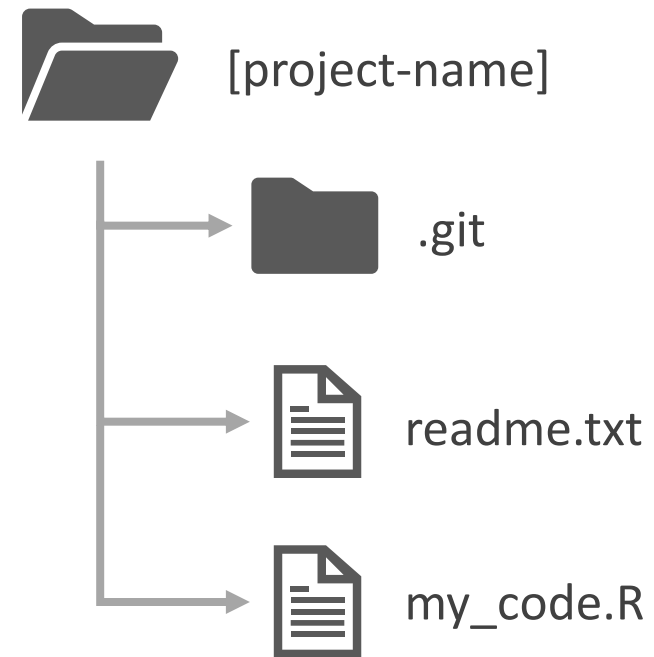
File Status Lifecycle

Stage untracked/modified file

```
git add [file-name]
```

Commit staged files

```
git commit -m "[message]"
```

Show **modification**

```
git diff HEAD
git diff [commit] [commit]
```

Stage all files

```
git add -A
```

HEAD
b3bd9b   b06ecf

master

committed

staged

modified

untracked

readme.txt   my_code.R
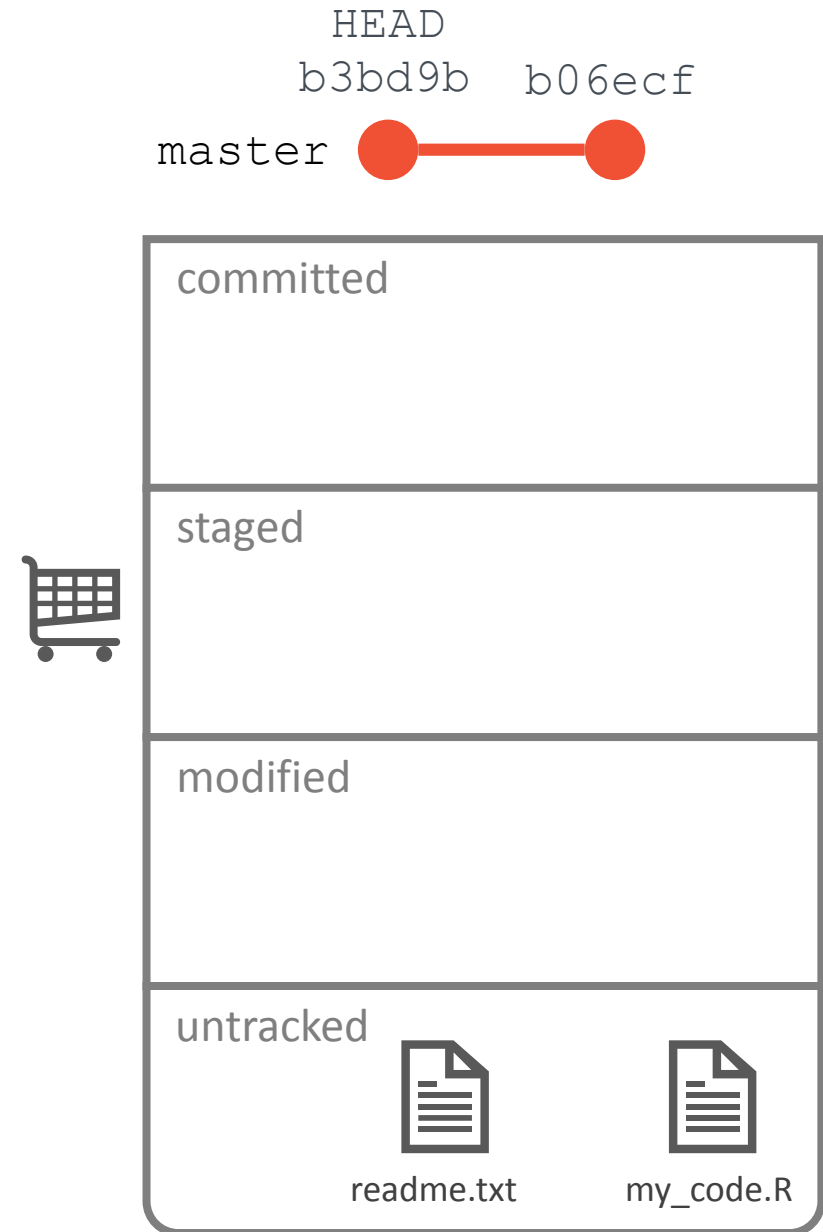
Log

Show **logged** commits

```
git log
```

Show command **options**

```
git help log
```

Show f**ormatted** log message and g**raph**

```
git log --graph --oneline
```

Restore previous version

```
git checkout [commit-hash]
```

Commit message

- Use imperative mode for summary line. Start with "Fix", "Add", instead of "Fixed", "Added".

- Leave the second line blank

- Line break the commit message (at about 72 characters)

- Don't end the summary line with a period, as it is a title

```
Short (50 chars or less) summary of changes

More detailed explanatory text, if necessary. Wrap it to
about 72 characters or so. In some contexts, the first
line is treated as the subject of an email and the rest
of the text as the body. The blank line separating the
summary from the body is critical (unless you omit the
body entirely); tools like rebase can get confused if you
run the two together.

Further paragraphs come after blank lines.

- Bullet points are okay, too

- Typically a hyphen or asterisk is used for the bullet,
  preceded by a single space, with blank lines in
  between, but conventions vary here
```
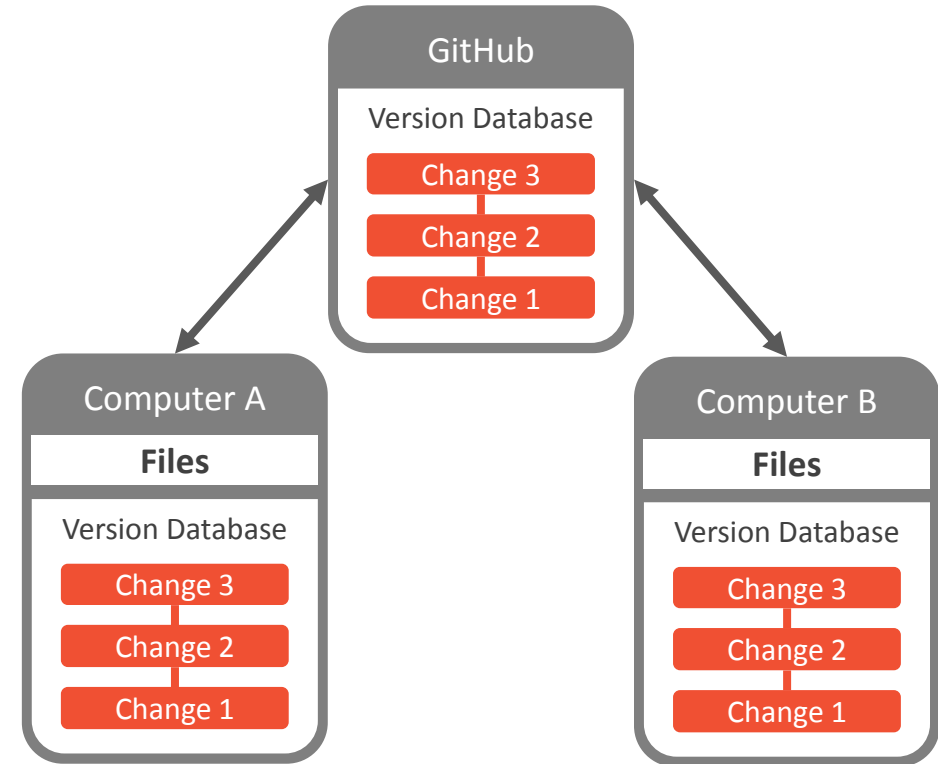
Remote Repositories

- **Bare repository** only contains version database

- **Working repository** generates working directory from version database

Distributed Version Control System

Remote Repositories

A project copy, which is hosted on a remote server: github.com/mi-erasmusmc/GitTutorial
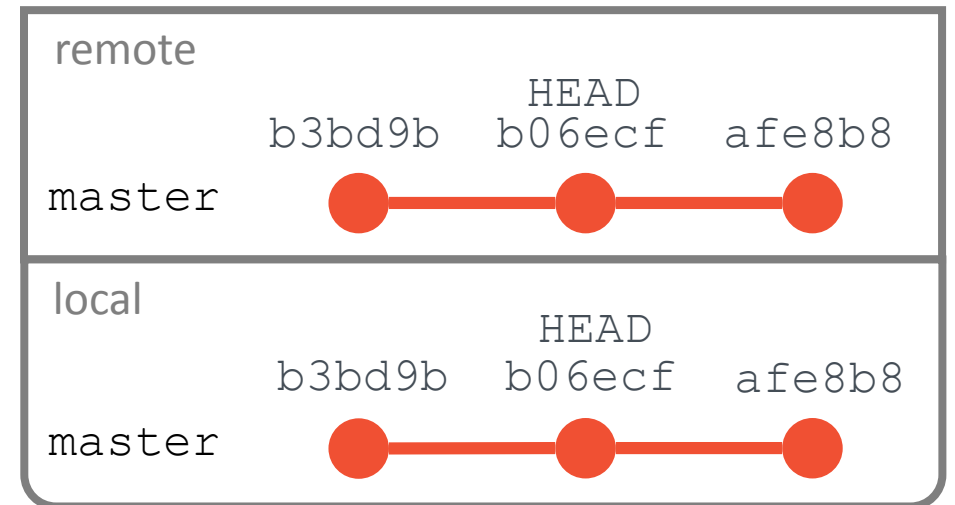
Clone a remote repository

```
git clone [url]
```

Lists remote repository **Information**

```
git remote –v

git branch -a
```

Push changes to remote repository

```
git pull

git push
```

Branches

Diverge from the main line of development to work independently.

List local branches

```
git branch
```
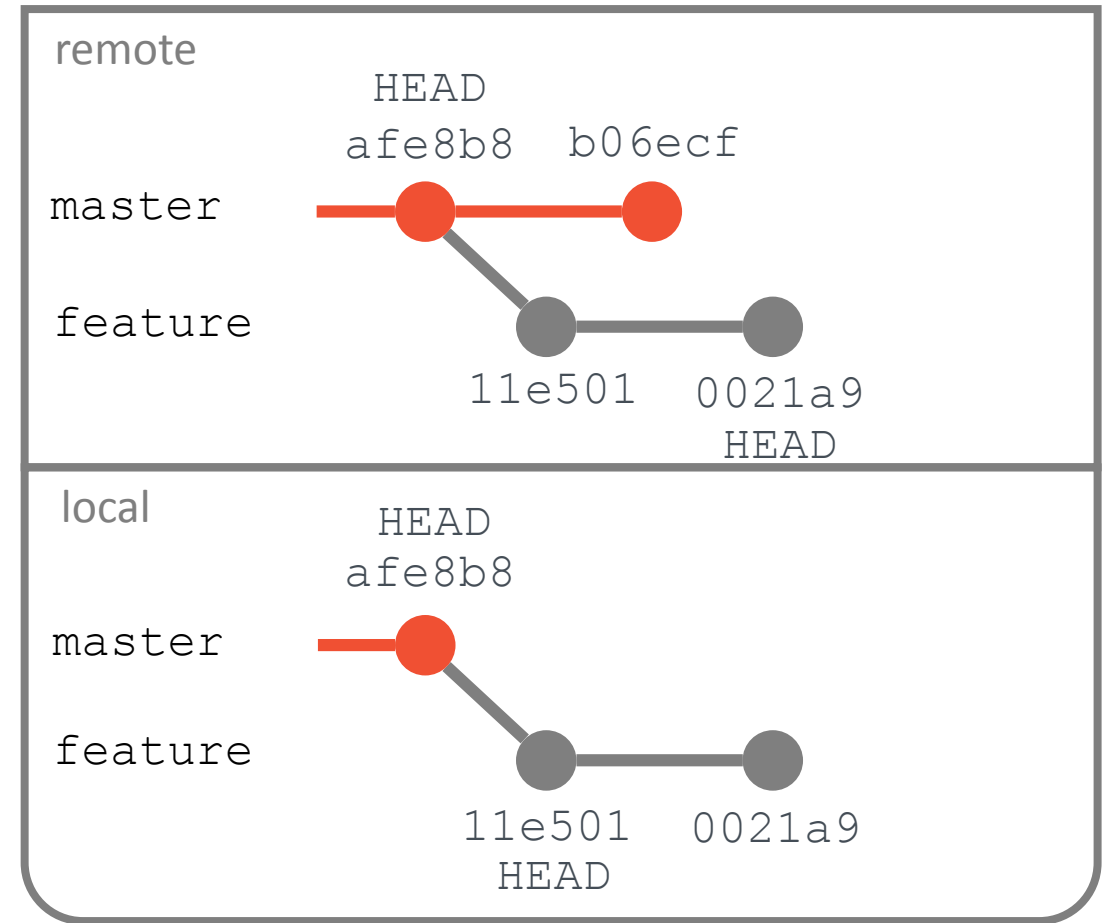
Create a branch

```
git branch [branch-name]
```

Enter a branch

```
git checkout [branch-name]
```

Push branch to remote repository

```
git push -u origin [branch-name]
```

Merging

Combine two lines of development after independent work is completed.

Pull **Master** branch changes

```
git checkout master
git pull
```
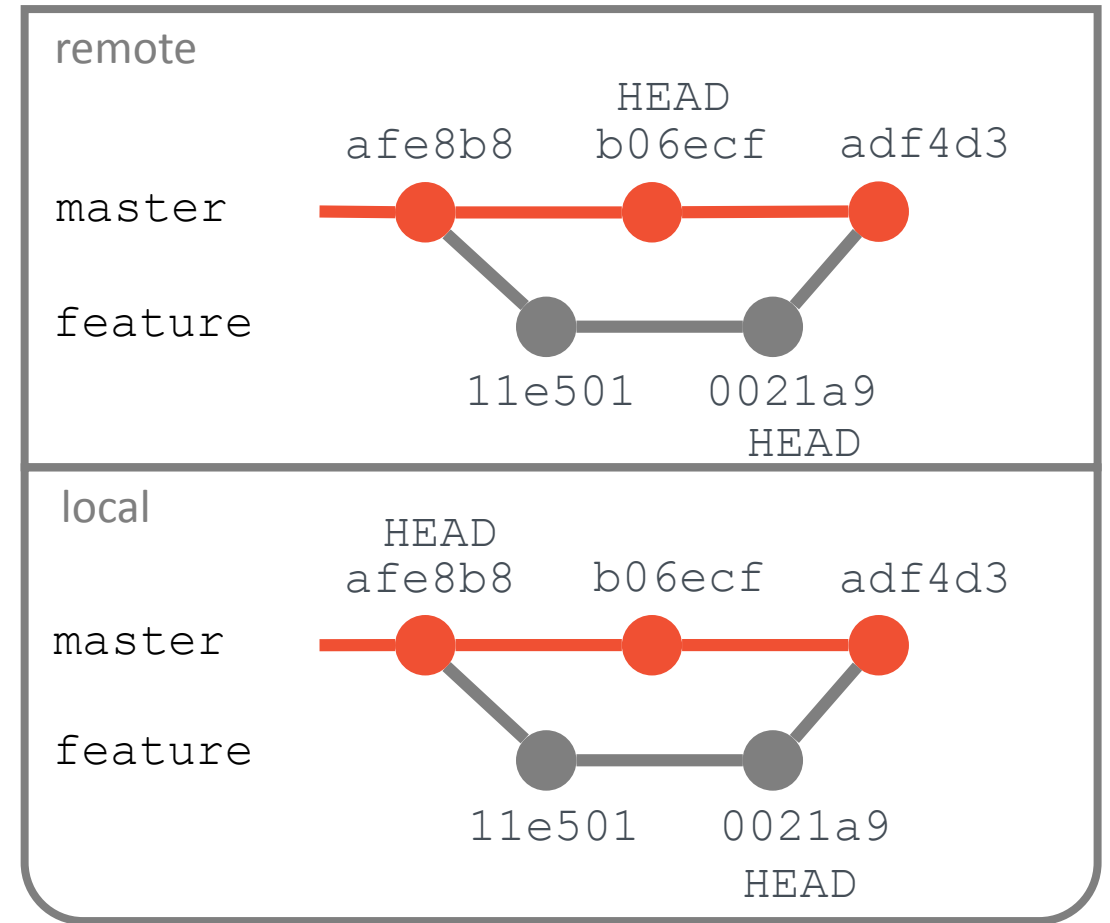
Show **Merged** branches

```
git branch --merged
```

Merge feature branch into master

```
git merge [branch-name]
```

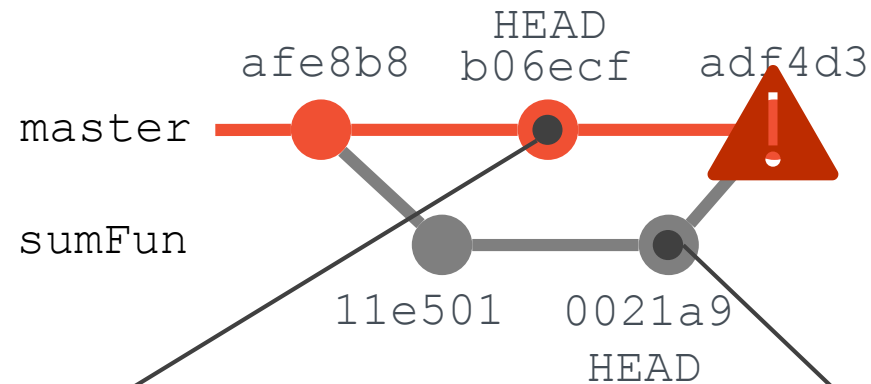## **Push** merge to remote repository

```
git push
```

Merge Conflicts

```
# Take sum of vector
sum <- function(x){
    sum <- 0
```

```
# Take sum of vector
if (any(is.na(x)) {
sum<<<<<<HEAD(x){
    sumtop ("x has NA values")
=======
    if(any(is.na(x)) {NA")
>>>>>>sumFun NA values")
    }

    for (val in x) {
        sum <- sum + val
    }
    sum
}
```

Amy
sum.R

HEAD
afe8b8   b06ecf   adf4d3

master   ●————————●————————⚠

sumFun

11e501   0021a9
          HEAD

git merge sumFun

Auto-merging sum.R

CONFLICT (content): merge conflict in sum.R

Automatic merge failed; fix conflicts and then commit the result.

git mergetool

```
# Take sum of vector
sum <- function(x){
    sum <- 0

    if (any(is.na(x)) {
        stop ("x contains NA")
    }

    for (val in x) {
        sum <- sum + val
    }
    sum
}
```

Bob
sum.R

Removing Branch

Show **Merged** branches

```
git branch --merged
```
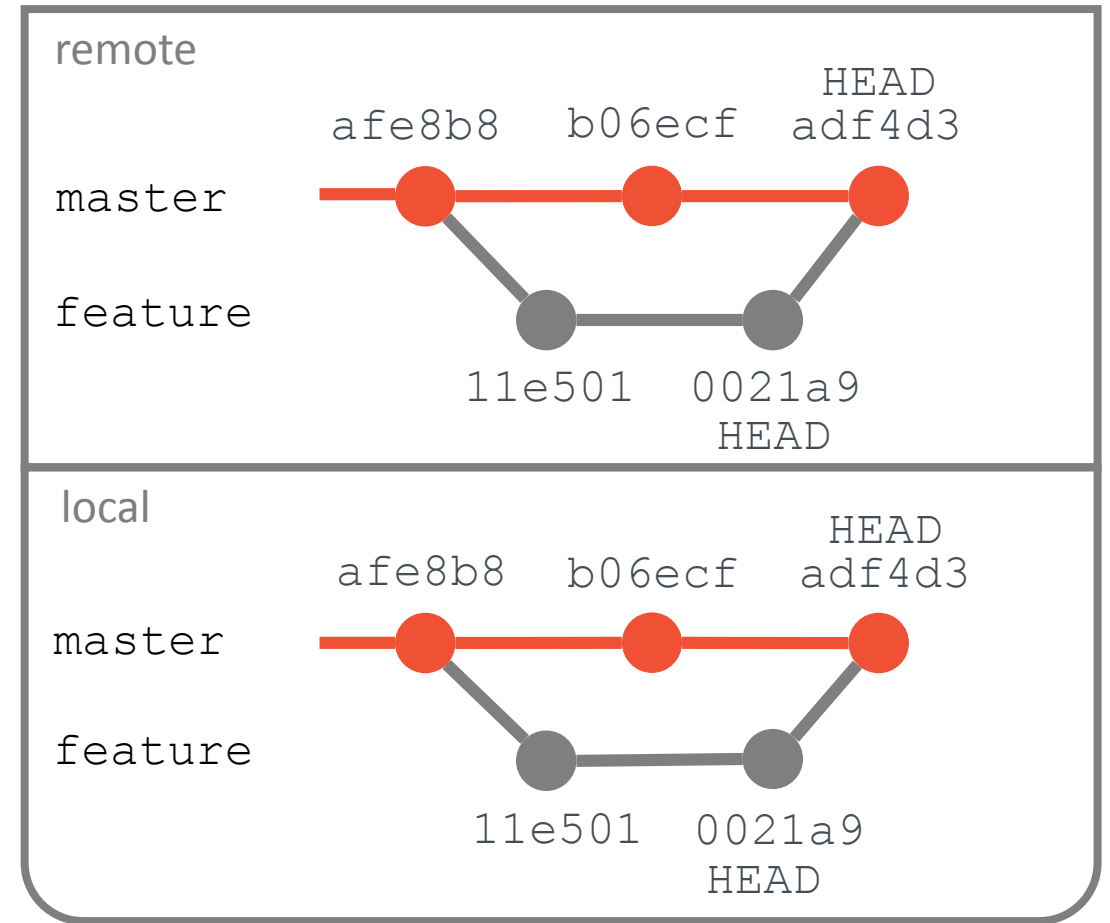
Remove branch

```
git branch –d [branch-name]
```

List **remote** branches

```
git branch -a
```

Remove branch from **remote** repository

```
git push origin –delete [branch-name]
```
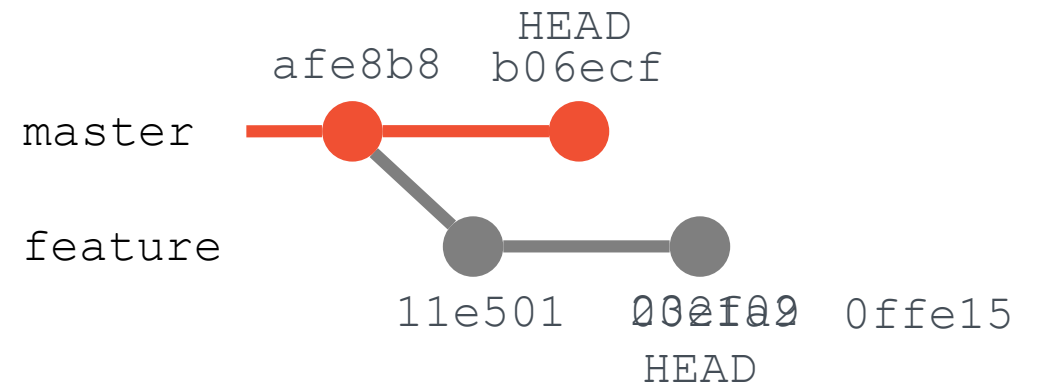
Rebase

Changes base of a branch from one commit to another. *Moved* branch is composed of entirely new commits.

Rebase branch

```
git checkout [branch-name]

git rebase master
```

Fast-Forward merge

```
git checkout master

git merge [branch-name]
```

Useful

Stage and commit all files

```
git commit –am “[message]”
```

Tag a commit

```
git tag
```

Set e**ditor** for commit messages - **default** `vim`
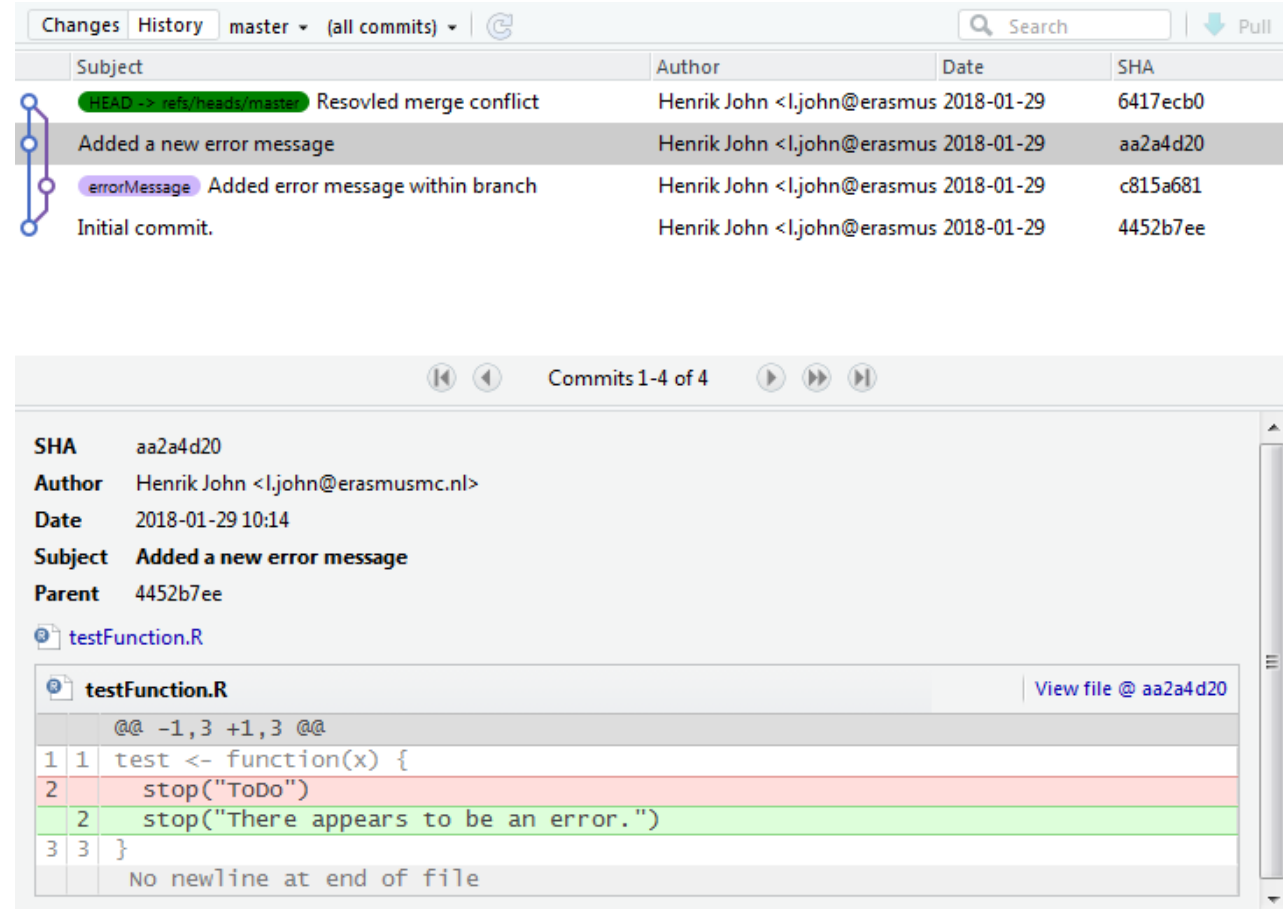
```
git config --global core.editor [editor]
```

 .gitignore

RStudio GUI

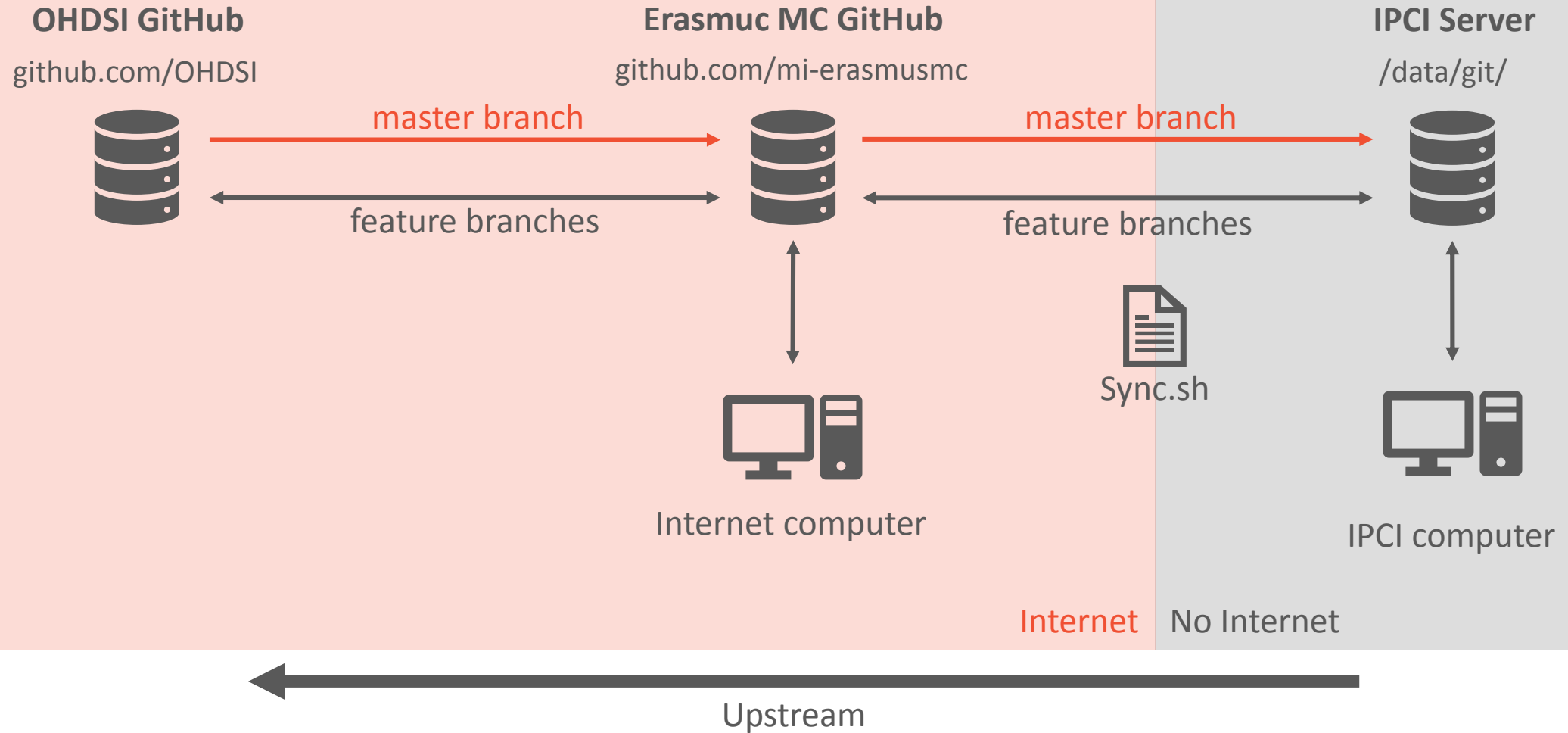- **Visualize** branches and changes
- **Merging** functionality <u>not</u> available

GitHub

- **Repositories** store all files of a particular project
- **Visualize** changelogs
- **Social network** infrastructure

Medical Informatics Workflow

Assignment

1. **Clone** the project from https://github.com/mi-erasmusmc/GitTutorial
2. Create a new **branch**
3. Within the new branch modify `participants.txt` by adding your first name to the list
4. **Commit** the change to your branch
5. **Merge** your branch into the master
6. **Push** the master to the remote

Documentation

- **Installation** files https://git-scm.com/downloads

- Official **documentation** https://git-scm.com/documentation

- **DataCamp** course https://www.datacamp.com/courses/introduction-to-git-for-data-science

- **Crash** course https://www.youtube.com/watch?v=SWYqp7iY_Tc

- **RStudio** course
  - Part 1: https://www.datacamp.com/courses/working-with-the-rstudio-ide-part-1
  - Part 2: https://www.datacamp.com/courses/working-with-the-rstudio-ide-part-2