L'écosystème Git : fonctionnement, outils, possibilités

Élise Maigné

L'écosystème Git

Git : fonctionnement, outils, possibilités

Elise Maigné

2022-02-18

git c'est quoi ?

Git

Un système de gestion de version (voir Différence avec SVN).

Qui permet :

- De sauvegarder ses codes (si serveur distant)
- De conserver l'historique des fichiers (qui a fait quoi ?)
- Visualiser les changements au cours du temps
- Travailler à plusieurs sur le même code
- Revenir en arrière, changer d'avis, tester une solution dans une branche séparée

Pourquoi je m'y suis pas mise avant ?

- 1ers mails sur git remontent à 2013 dans mon équipe et 2014 groupe RR.
- J'avais vraiment du mal à comprendre comment ça marchait et ce que je pouvais faire.
- J'étais perdue au milieu de tous les termes "git/push/github/tortoise".
- Différence de langages entre info/stat

"Elise j'ai mis en place un SVN, c'est facile tu synchronyse avec tortoise et tu push tes modifs"

git et son écosystème

Ecosystème 1

un projet (de code) que l'on veut versionner

Ecosystème 2 (facultative)

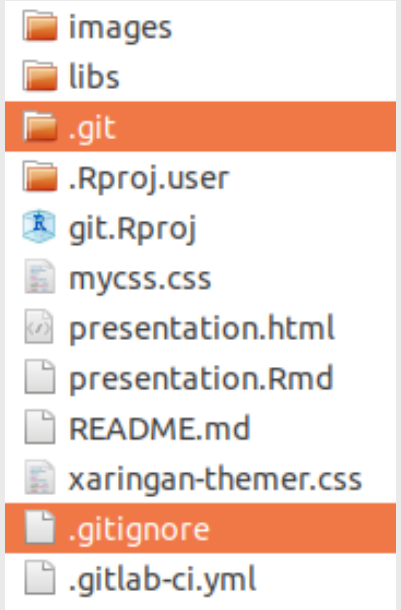
choisir un endroit pour déposer son code

Ecosystème 3 (facultative sur mac et linux)

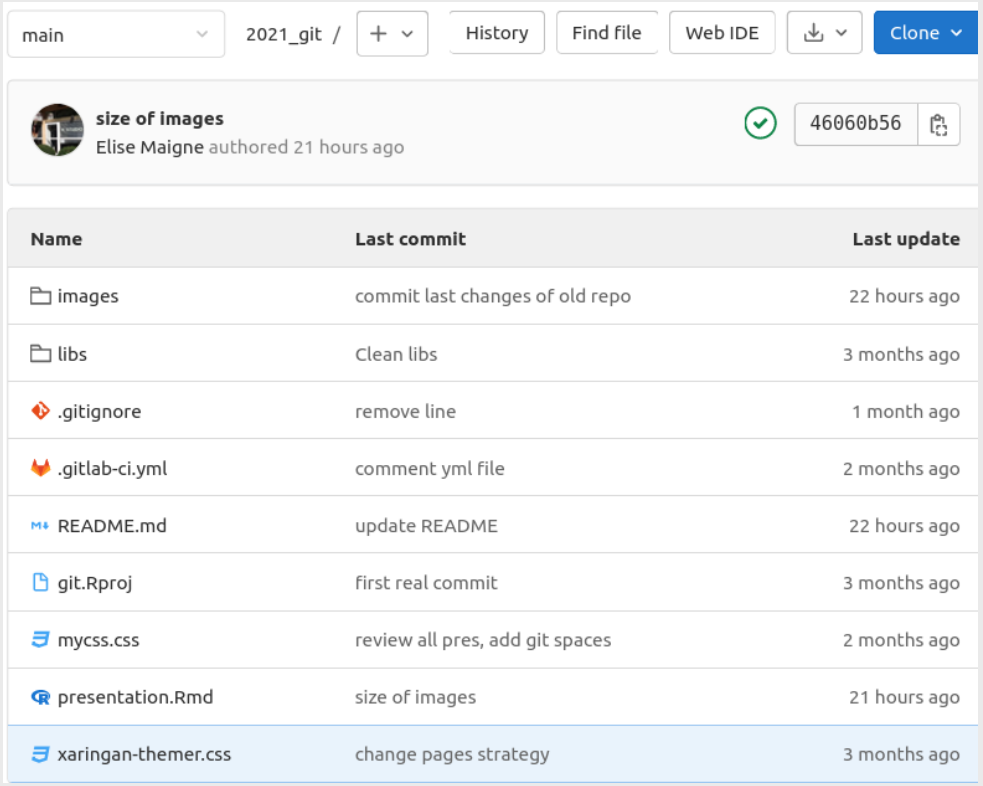
un logiciel plus sympa que la ligne de commande

A quoi ressemble un projet versionné ?

Sur ma machine :



Sur forgemia.inra.fr :



Git fonctionnement

Quelques commandes de base

Initialisation

Prérequis : j'ai mis ma clé ssh sur un serveur distant

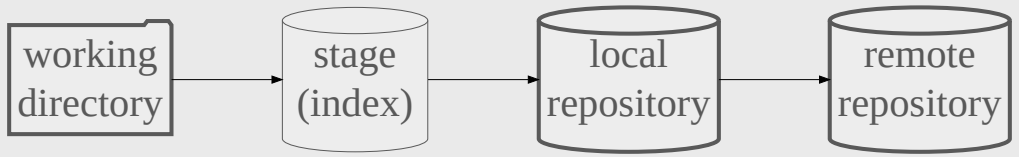
Je démarre un projet sur la forge

1. Création d'un projet sur une forge (avec README)
2. Clone de ce projet sur sa machine en local

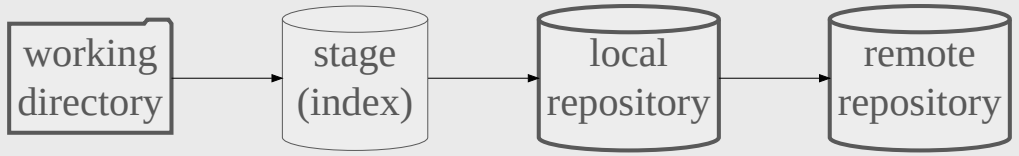
```
git clone git@forgemia.inra.fr:elise.maigne/20
```

J'ai déjà un dossier en local

1. `git init`
2. Création d'un projet sur une forge (sans README)
3. "Push an existing repository"

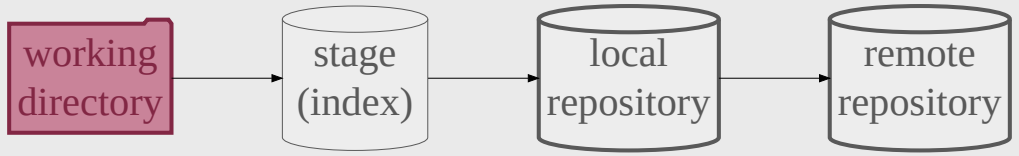


Les "espaces" git



- **working directory**
- **stage**
- **local repository**
- **remote repository**

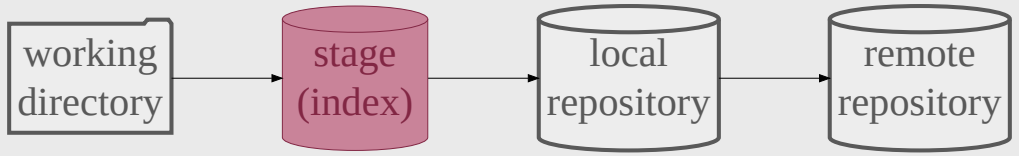
Les "espaces" git



- **working directory**
- **stage**
- **local repository**
- **remote repository**

Mon espace de travail classique, sur ma machine. Là où je code.

Les "espaces" git

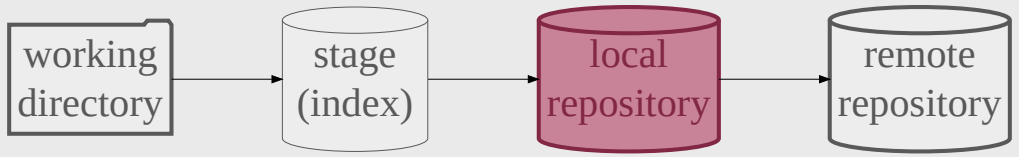


- **working directory**
- **stage**
- **local repository**
- **remote repository**

Aussi appelé **index** ou **zone de stagging**.

Un espace de transit sur lequel sont observées les modifications de fichier.

Les "espaces" git

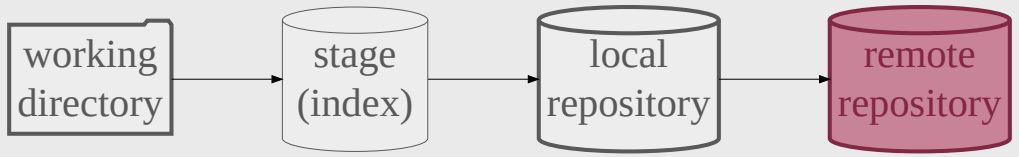


- **working directory**
- **stage**
- **local repository**
- **remote repository**

Le dépôt de version local, sur ma machine.

Enregistre tous les changements (commits).

Les "espaces" git



- **working directory**
- **stage**
- **local repository**
- **remote repository**

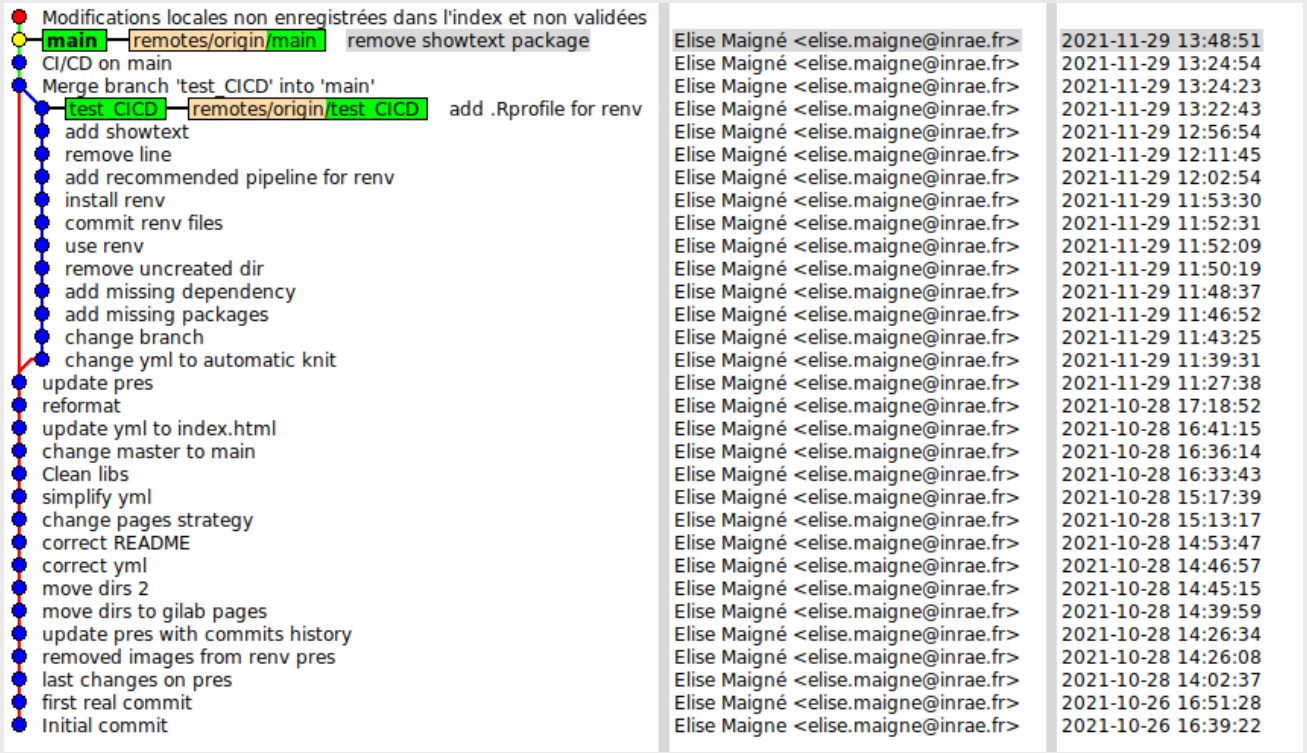
Le dépôt de version distant, sur une forge.

Enregistre tous les changements (commits) et permet de partager son dépôt.

Un projet versionné = une succession de commits.

Exemple sur forgemia : https://forgemia.inra.fr/elise.maigne/2021_git/-/commits/main

Exemple avec gitk



Un commit = un point d'étape

SHA 966eb4beba5fb200b05fe1f248093e520c3ed671
Author Elise Maigné <elise.maigne@inrae.fr>
Date 2021-11-29 12:48
Subject **remove showtext package**
Parent 152cadfa5aa192258bd207cfe1025e2f4481c05d

.gitignore
.gitlab-ci.yml
presentation.Rmd

.gitignore View file @ 966eb4be

```
@@ -2,3 +2,4 @@
2 2 .Rhistory
3 3 .RData
4 4 .Ruserdata
  5 presentation.html
```

.gitlab-ci.yml View file @ 966eb4be

```
@@ -13,7 +13,6 @@ cache:
13 13
14 14 before_script:
15 15   - Rscript -e "if (!requireNamespace('renv', quietly = TRUE)) install.packages('renv')"
16    -   - Rscript -e "install.packages('showtext')"
17 16   - Rscript -e "renv::restore()"
18 17
19 18 pages:
```

Un commit = un point d'étape

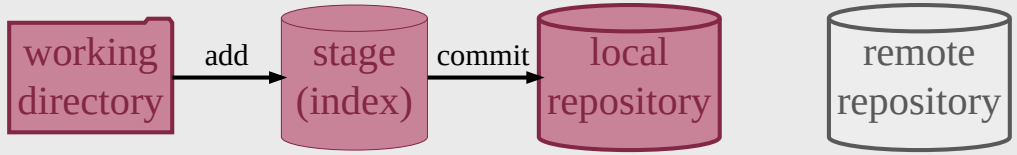
1. On indique à git qu'il doit suivre ces fichiers et on intègre les modifications au prochain commit `git add`

```
git add .
```

```
git add nomFichier.R
```

2. On fait un commit avec `git commit`

```
git commit -m "J'explique ce que j'ai fais comme modif"
```



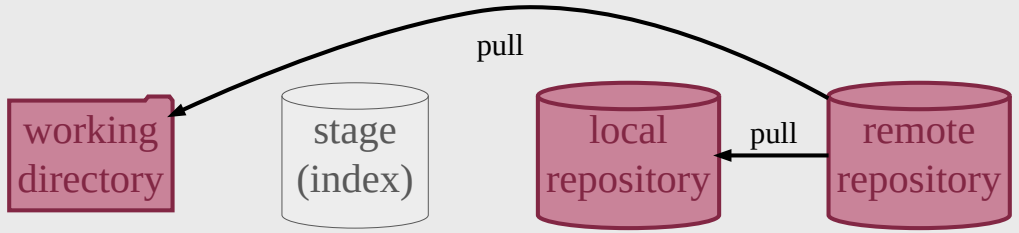
Et le serveur distant ?

Envoi des derniers commits au serveur distant

```
git push
```

Récupération de l'état du du serveur distant

```
git pull
```



Exemple de processus "simple" de travail avec git

1. Un projet versionné sur un serveur distant

2. Je travaille comme d'habitude sur mes fichiers

3. De temps en temps = je marque un point d'étape de mon travail :

```
git add .
git commit -m "J'explique en 2 mots ce que j'ai fait"
git push
```

4. Si travail à plusieurs : je récupère les modifications des autres (avant de me remettre à travailler)

```
git pull
```

Les fichiers git

- Un dossier `.git` : contient tous les fichiers de gestion de version.
- Un fichier `.gitignore` : permet d'exclure certains fichiers ou dossiers des commits.

Exemple de contenu d'un `.gitignore`:

```
.Rproj.user
.Rhistory
.RData
.Ruserdata
/data
```

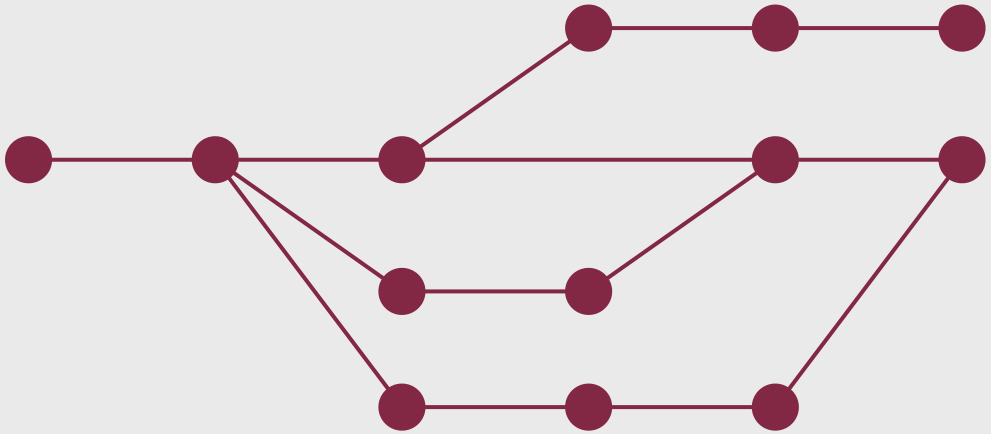
En particulier on ne commite pas les données.

Les branches

La branche par défaut est **main** (feu **master**).

Il est possible de créer de nouvelles branches, pour tester une autre direction, ajouter une fonctionalité.

Projet d'application à plusieurs, 1 branche = 1 fonctionalité, plusieurs fonctionalités en même temps.



Développement d'un package R : une branche correspond à une modification majeure du code, faite à plusieurs.

Les branches

Création d'une branche

```
git branch mabranche
```

Changer de branche

```
git checkout mabranche
git checkout monautrebranche
```

On ne peut changer de branche uniquement si toutes les modifications (des fichiers suivis) sont commitées.

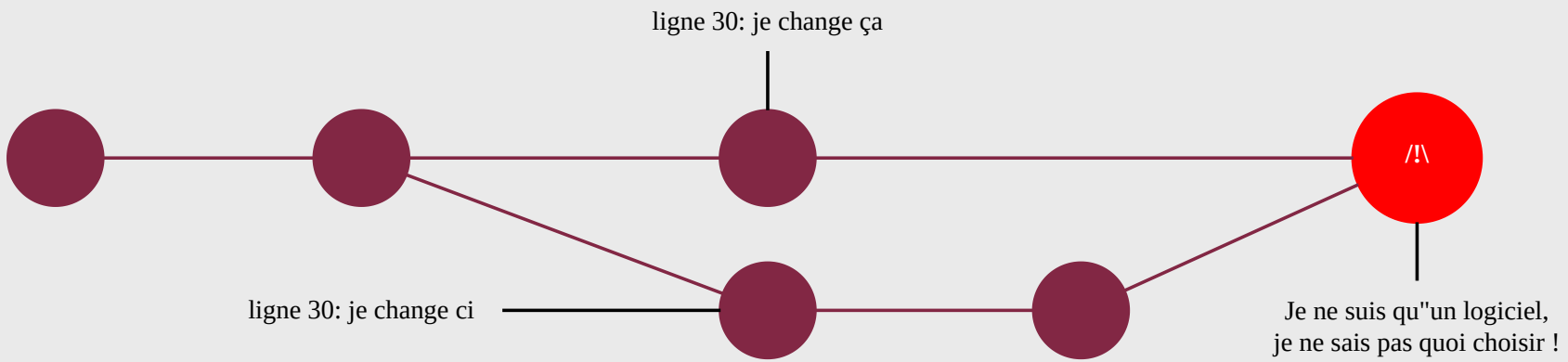
Les branches

Fusionner des branches

```
git checkout main   # Je me place sur main
git merge mabranche   # J'intègre les modifications qu'il y a sur "mabranche"
```

Les conflits

Si un fichier est modifié 2 fois au même endroit. Il faut choisir entre les 2 versions pour pouvoir finaliser le merge



Les branches

A quoi ressemble un conflit ?

```
<<<<<<< HEAD:fichier.R
J'avais mis ça avant dans une branche
======
Et maintenant j'ai ça, je ne suis qu'un logiciel je ne sais pas quoi faire
>>>>>>> iss53:fichier.R
```

Merge request/pull request (sur une forge)

Crée une demande de rajout de code.

Superviser les "merge" à l'intérieur d'un projet ou permet à une personne extérieure au projet de contribuer.

- Dicussion
- Gestion des conflits
- Assigner à quelqu'un

Vocabulaire : `merge request` chez GitLab VS `pull request` chez GitHub

Les forges à votre disposition

- GitHuB
- GitLab
- pour les INRAE : forgemia.inra.fr (instance GitLab)
- communauté RENATER : sourcesup.renater.fr (instance FusionForge, avec git)
- communauté mathématique française : https://plmlab.math.cnrs.fr (instance GitLab)

Possibilités (pour des statisticiens)

- **Publication, Partage du code**
- **Branches**
- **GitLab pages, GitHub pages**
- **CI/CD (integration continue)**
- **Enseignement**

Code bien identifié, accessible via une url.

Ne pas oublier le README à la racine !

Transparence et visibilité.

Possibilités (pour des statisticiens)

- **Publication, partage du code**
- **Branches**
- **GitLab pages, GitHub pages**
- **CI/CD (integration continue)**
- **Enseignement**

Se servir des branches pour tester :

- une nouvelle méthode,
- une hypothèse différente,
- un nouveau jeu de données

Possibilités (pour des statisticiens)

- **Publication, partage du code**
- **Branches**
- **GitLab pages, GitHub pages**
- **CI/CD (integration continue)**
- **Enseignement**

Faire des sites webs très facilement avec :

- un code qui génère un html (ou un html dans mon dépôt)
- et juste un fichier yml à la racine du dépôt

(exemple : https://elisemaigne.pages.mia.inra.fr/2021_git/index.html)

Exemple de fichier yml (nom de fichier = ".gitlab-ci.yml") :

```
pages:
  stage: deploy
  # Je place le nécessaire dans un dossier public/
  script:
  - mkdir public
  - cp -r presentation/* public
  artifacts:
    paths:
    - public
  # Je fais tout ça uniquement sur la branche main
  only:
  - main
```

Possibilités (pour des statisticiens)

- **Publication, partage du code**
- **Branches**
- **GitLab pages, GitHub pages**
- **CI/CD (integration continue)**

  Compiler ses rapports automatiquement.

  Faire des tests automatiques (ex. `R cmd CHECK`, lancer des tests unitaires, ...).

  Attention, c'est fait à chaque "push" --> Est-ce toujours utile ?
- **Enseignement**

Possibilités (pour des statisticiens)

- **Publication, partage du code**
- **Branches**
- **GitLab pages, GitHub pages**
- **CI/CD (integration continue)**
- **Enseignement**

  Un projet "principal" forké (`git fork`) par les étudiants qui partent de la même copie et peuvent faire leurs propres modifications, indépendemment les un des autres.

Vocabulaire

Au démarrage

- `git clone`
- `git fork`
- `git init`

Gérer les modifications de code

- **`git add` + `git commit -m "Message'`**
- **`git push`**
- **`git pull`**

Visualiser

- `git diff`
- `git blame`

Savoir où on en est

- **`git status`**

Branches

- `git branch`
- `git checkout`
- `git merge`
- `git rebase`

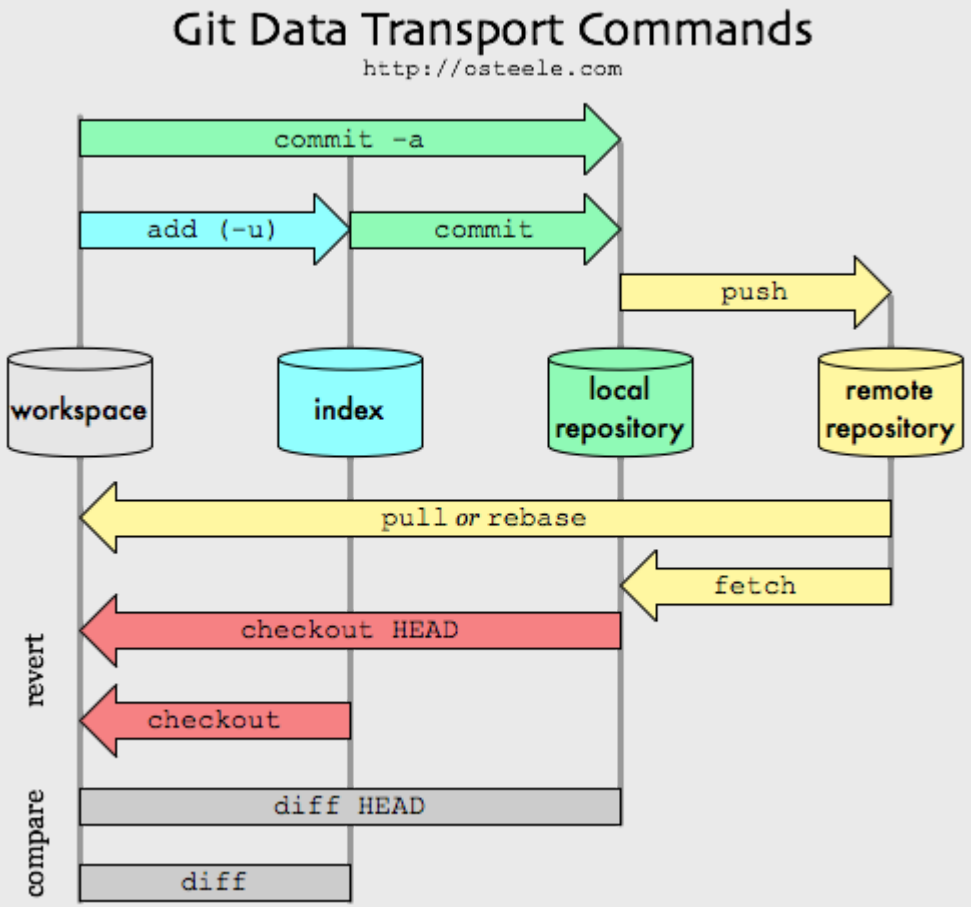
Revenir en arrière

- `git revert`

Un mot que l'on risque de croiser en travaillant avec git

- `HEAD` = pointe vers un commit (= un état du dépôt, classiquement le dernier commit de la branche sur laquelle on est)

Des points de repère



©Oliver Steele, https://blog.osteele.com/2008/05/my-git-workflow/

des questions ?

Lien vers le dépôt de cette présentation : https://forgemia.inra.fr/elisemaigne/2021_git/

Références

- https://perso.liris.cnrs.fr/pierre-antoine.champin/enseignement/intro-git/
- http://rogerdudler.github.io/git-guide/index.fr.html
- Doc de la forgemia : https://forgemia.inra.fr/adminforgemia/doc-public/-/wikis/GIT-sous-Windows
- branches git en graphviz : https://correl.phoenixinquis.net/2015/07/12/git-graphs.html
- Des jolies animations : https://www.miximum.fr/blog/enfin-comprendre-git/
- Comprendre l'index (stage) : https://medium.com/hackernoon/understanding-git-index-4821a0765cf

Des cours en ligne

- Apprendre à utiliser Git et GitHub - 2020
- Utiliser GIT avec R, INSEE, 2021
- Tutoriel GitLab, 2018

Bonus - Différence SVN/Git ()

SVN

SVN est un système de contrôle de version **centralisé**. Vous avez un serveur qui contient votre référentiel de code. Vous pouvez en extraire du code sur votre ordinateur local, y apporter des modifications locales, puis les renvoyer dans le référentiel central.

Votre copie du code correspond généralement à ce que vous avez extrait et à la dernière version. De nombreuses opérations nécessiteront une connexion réseau au référentiel central.

Git

Git est un système de contrôle de version **décentralisé**. Chaque participant a un clone de l'ensemble du référentiel. Il est utilisé pour suivre les changements dans le code source.

La plupart des opérations ne nécessitent pas de connexion réseau, car elles ne travaillent que sur votre clone du référentiel.

Source : https://waytolearnx.com/2019/03/difference-entre-git-et-svn.html

Regarder les différences entre le local et un autre état

```
git diff
```

395		#### Publication/partage
	410	#### Publication/partage du code
396	411	
397	412	#### gitlab pages
398	413	Faire des sites webs très facilement ! avec juste un fichier yml à la racine du dépôt
399	414	
400	415	Exemple : j'ai un projet qui crée un fichier html à l'aide de Rmarkdown. --> Si j'active gitlab pages je peux en faire un site web.
401		(exemple : TODO)
	416	(exemple : https://elise.maigne.pages.mia.inra.fr/2021_git/index.html)
402	417	
403	418	--
404	419	

Exemple de fichier yml (nom de fichier = .gitlab-ci.yml)

```
pages:
  stage: deploy
  # Je place le nécessaire dans un dossier public/
  script:
  - mkdir public
  - cp -r presentation/* public
  artifacts:
    paths:
    - public
  # Je fais tout ça uniquement sur la branche main
  only:
  - main
```

Exemple de fichier yml (nom de fichier = .gitlab-ci.yml)

```yaml
# J'appelle une image docker qui contient R, rmarkdown, ... https://hub.docker.com/r/rocker/verse
image: rocker/verse:4.0.0

# J'installe les packages qui ne sont pas dans l'image rocker/verse
before_script:
  - Rscript -e "install.packages('showtext')"
  - Rscript -e "install.packages('DiagrammeR')"
  - Rscript -e "install.packages('xaringanthemer')"
  - Rscript -e "install.packages('widgetframe')"

pages:
  stage: deploy
  # Je compile mon document et je place le nécessaire dans un dossier public/
  script:
  - Rscript -e "rmarkdown::render('presentation.Rmd', output_file = 'index.html')"
  - mkdir public
  - cp index.html xaringan-themer.css mycss.css public/
  - cp -r images/ libs/ public/
  artifacts:
    paths:
    - public
  # Je fais tout ça uniquement sur la branche main
  only:
  - main
  interruptible: true
```