

Module TC4

Gestion des versions – Versioning

Git - *Fonctionnalités avancées*





Chronologie des étapes de traitement d'un dépôt local

Sur le plan conceptuel, on peut dire que :

- En utilisant la commande **git add**, on demande à Git de suivre certains fichiers, c'est-à-dire de prendre en charge la gestion de l'évolution de ces fichiers.
- Par la commande **git commit**, on demande à Git d'enregistrer un **snapshot**, c'est-à-dire une photographie instantanée de l'état des fichiers suivis (plus exactement, des modifications qui ont été placées dans l'index, mais il s'agit là en fait d'une facilité),
 - Une suite de "commits" constitue un *historique*.

La commande git commit fournit le résultat suivant : le résultat d'une action de commit, représenté par un hash SHA-1 :

b866ab9738b880d9f110b6767c1adbd575b0cba

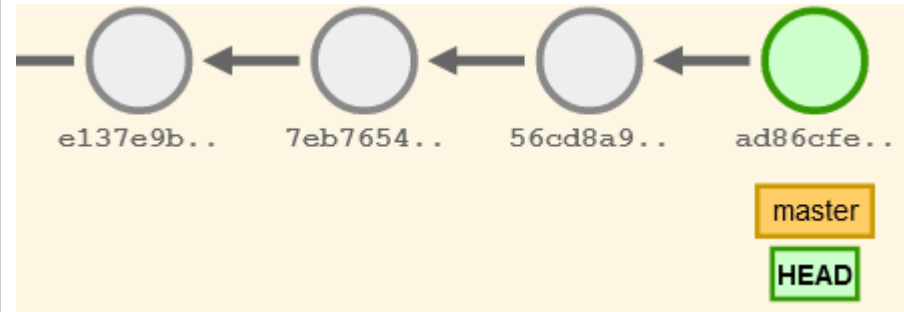
Votre répertoire de travail se compose de :

- L'ensemble des fichiers *suivis*, **dans leur version correspondant à UN SNAPSHOT COURANT** (un commit)
- Les modifications existant sur ces fichiers par rapport à cette version spécifique
- Des fichiers non suivis (pour lesquels on n'a pas effectué de git add)

> Manipulation de la référence HEAD

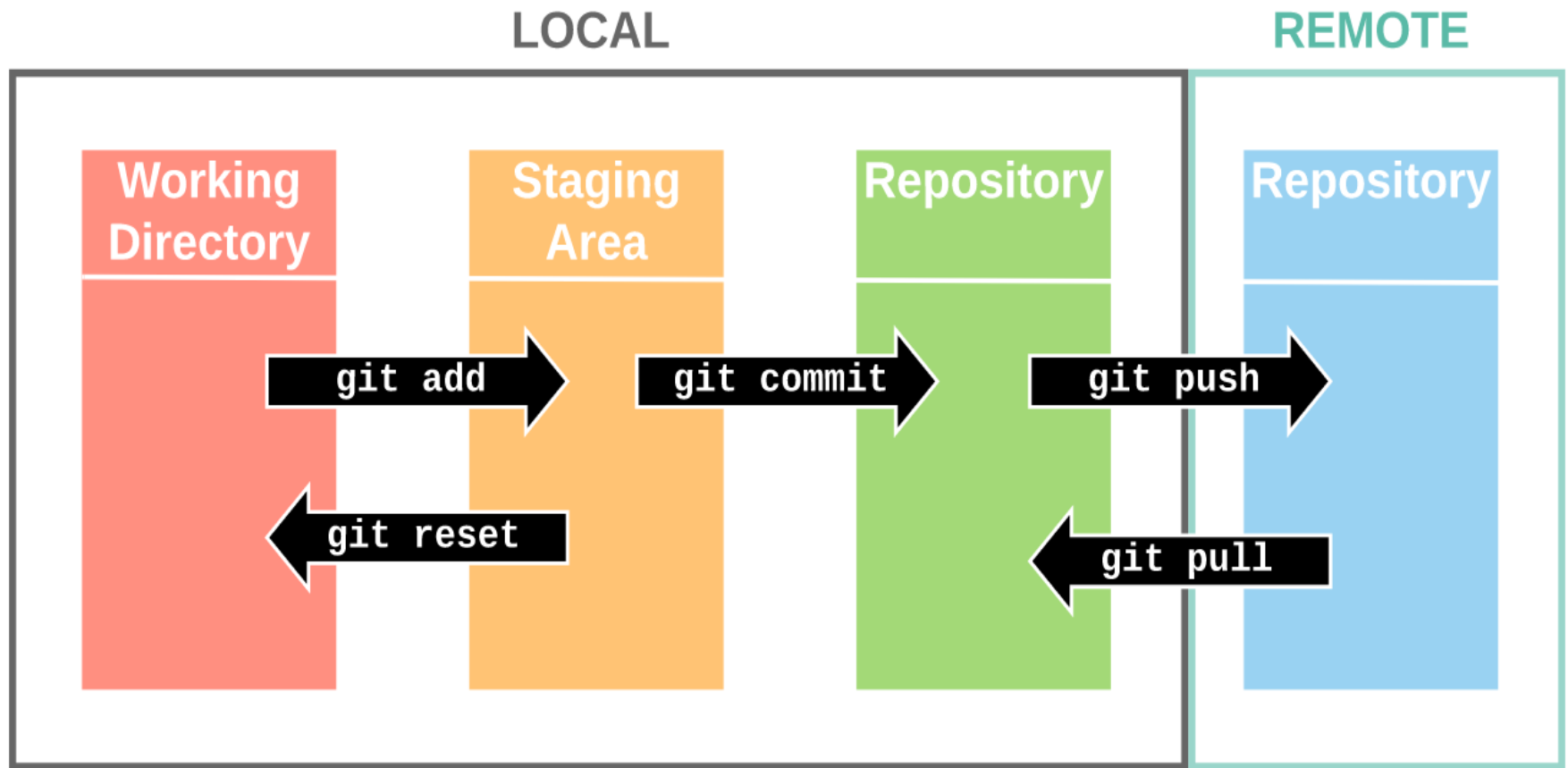
La commande `git commit` effectue deux actions :

- 1) Enregistre un *snapshot* ayant un lien de parenté avec le commit référencé par HEAD
- 2) Déplace la référence HEAD et la référence master (main) sur ce nouveau commit.

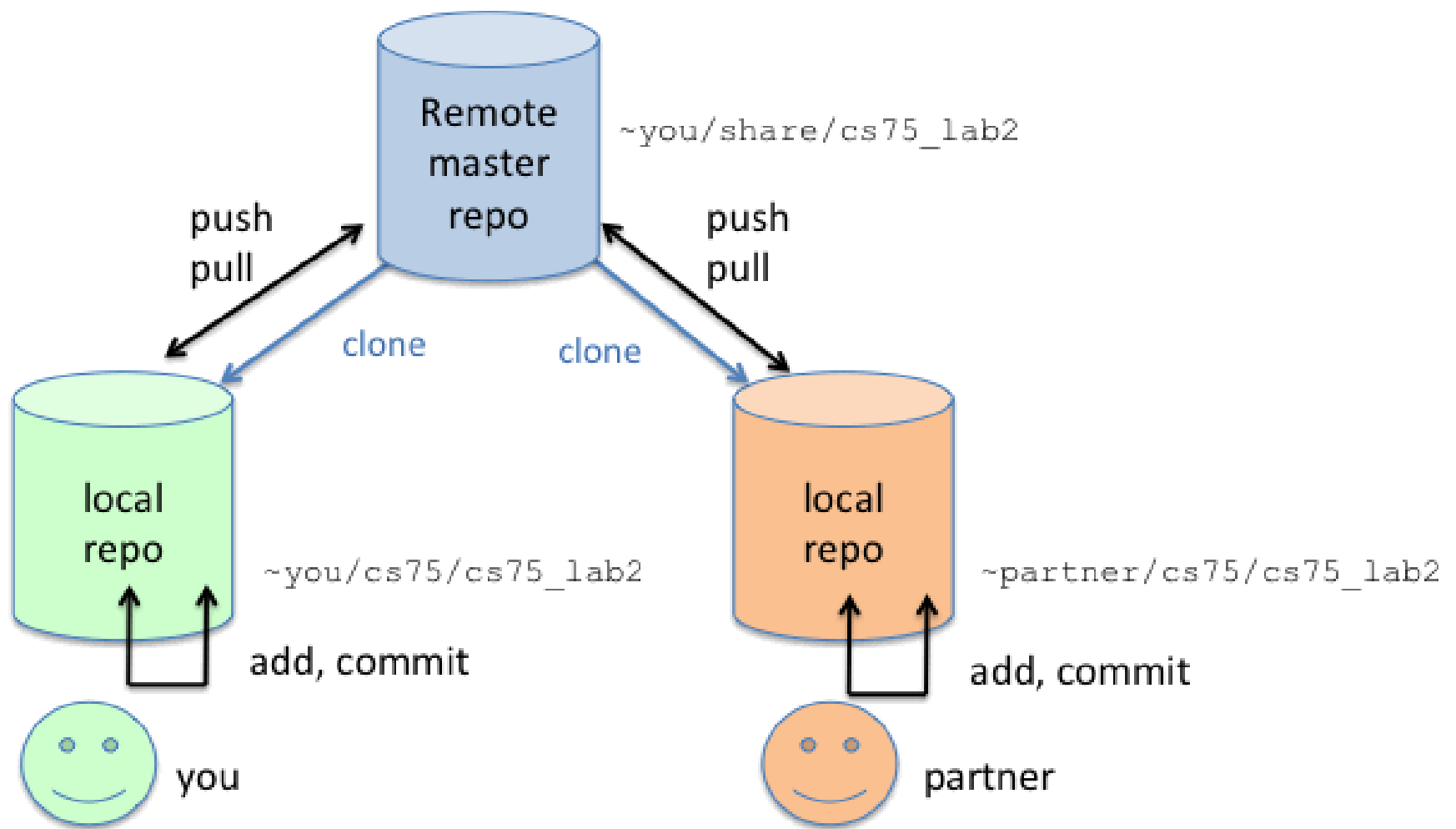


```
PS F:\2020-2021\LDV\CLASSES\BTSSIO\SEMESTRE_2\SI012\TC4\SEANCE_1-1\TPPHP> git log --oneline
aaa7a7e (HEAD -> main, origin/main, origin/develop, develop) branche développement 2
a73dd25 Nouvelle branche développement
f6610e9 Modification fichiers P00
9f56e49 Ajout dossier P00
71308a7 Ajout Partiel
a332280 Modif Tableau2
4e5db54 Modif tableau1
b866ab9 Ajout creationSpec
0c74f06 Ajout des fchiers
2fcd9c9 First Commit
PS F:\2020-2021\LDV\CLASSES\BTSSIO\SEMESTRE_2\SI012\TC4\SEANCE_1-1\TPPHP>
```

> Manipulation de la référence HEAD



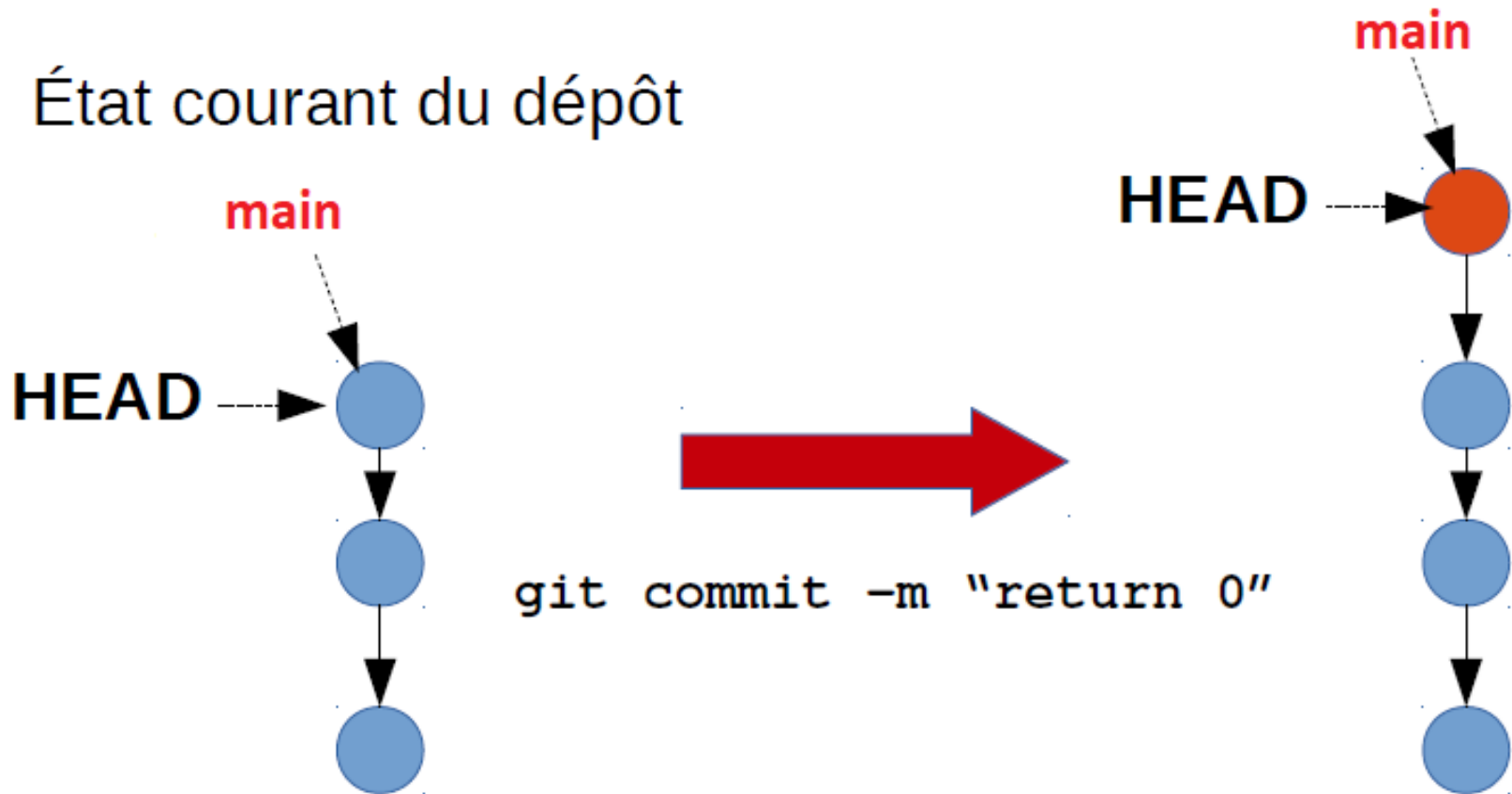
> Manipulation de la référence HEAD





Chronologie des étapes de traitement d'un dépôt local

État courant du dépôt





Annulation d'actions dans git

Si on valide une version (commit), puis on réalise qu'on a oublié d'indexer les modifications d'un fichier qu'on veut ajouter à ce commit, on utilisera utiliser les commandes suivantes :

```
git commit -m 'validation initiale'  
git add fichier_oublie  
git commit --amend
```

Pour revenir à une version précédente, procéder comme suit :

- Récupérer l'id du commit concerné (Chaque commit possède un id), la commande :
`git log --oneline`
- Lancer la commande suivante pour revenir à la version précédente :
`git checkout commitId`
- Pour revenir à la version initiale, lancer la commande :
`git checkout main`

Récupère la dernière version commitée d'un fichier (Modifications perdues)

```
git checkout fichier
```



Annulation d'actions

- Lister l'historique des fichiers indexés
`git ls-files --stage`
- Retirer un fichier de l'index (index), suite à une commande git add :
`git reset`
- Suppression définitive de toutes les modifications non commitées.
`git reset --hard`

Application

Lors d'une indexation et du commit de tous les fichiers du répertoire de travail, une erreur a glissé sur le fichier helpme.doc, qui ne devrait pas être indexé,

Donner les commandes git nécessaires pour résoudre ce problème.



Annulation d'actions

//Effectuer un commit par erreur

```
git add .  
git commit -m "Premier commit"
```

//Phase annulation

```
git reset --soft HEAD~1      //On ramène le head d'un pas vers l'arrière  
git reset tableau1.php       //Sort le fichier indiqué du stage
```

```
echo 'tableau1.php' >> .gitignore //Ignorer ce fichier la prochaine fois  
git add .gitignore
```

//Refaire le commit

```
git commit -m "Ajout de la clé publique"
```



Outil gitk

gitk: gitk-demo

File Edit View Help

master remotes/origin/master third commit Tony Stark <tony@stark.com> 2010-09-03 10:5
second commit Tony Stark <tony@stark.com> 2010-09-03 10:5
Initial commit Tony Stark <tony@stark.com> 2010-09-03 10:4

3. commit message

4. local branch "master"

5. remote branch master on origin

6. current commit (HEAD)

2. commit author

8. commit SHA of selected

SHA1 ID: 3d024dd9e4a83d8c6a9a143a68b75d4b872115a6 Row 2 / 3

Find next prev commit containing: Exact All fields

Search

Diff Old version New version Lines of context: 3 Ignore space cha

Committer: Tony Stark <tony@stark.com> 2010-09-03 10:50:28
Parent: b094e60a4888cefd57ce9316084b6e09f7cc3ea8 (Initial commit)
Child: bf37c64e79b9804aee541f590ccdab0466e01334 (third commit)
Branches: master, remotes/origin/master
Follows:
Precedes:

second commit

10. file changes in diff format

fruits.txt

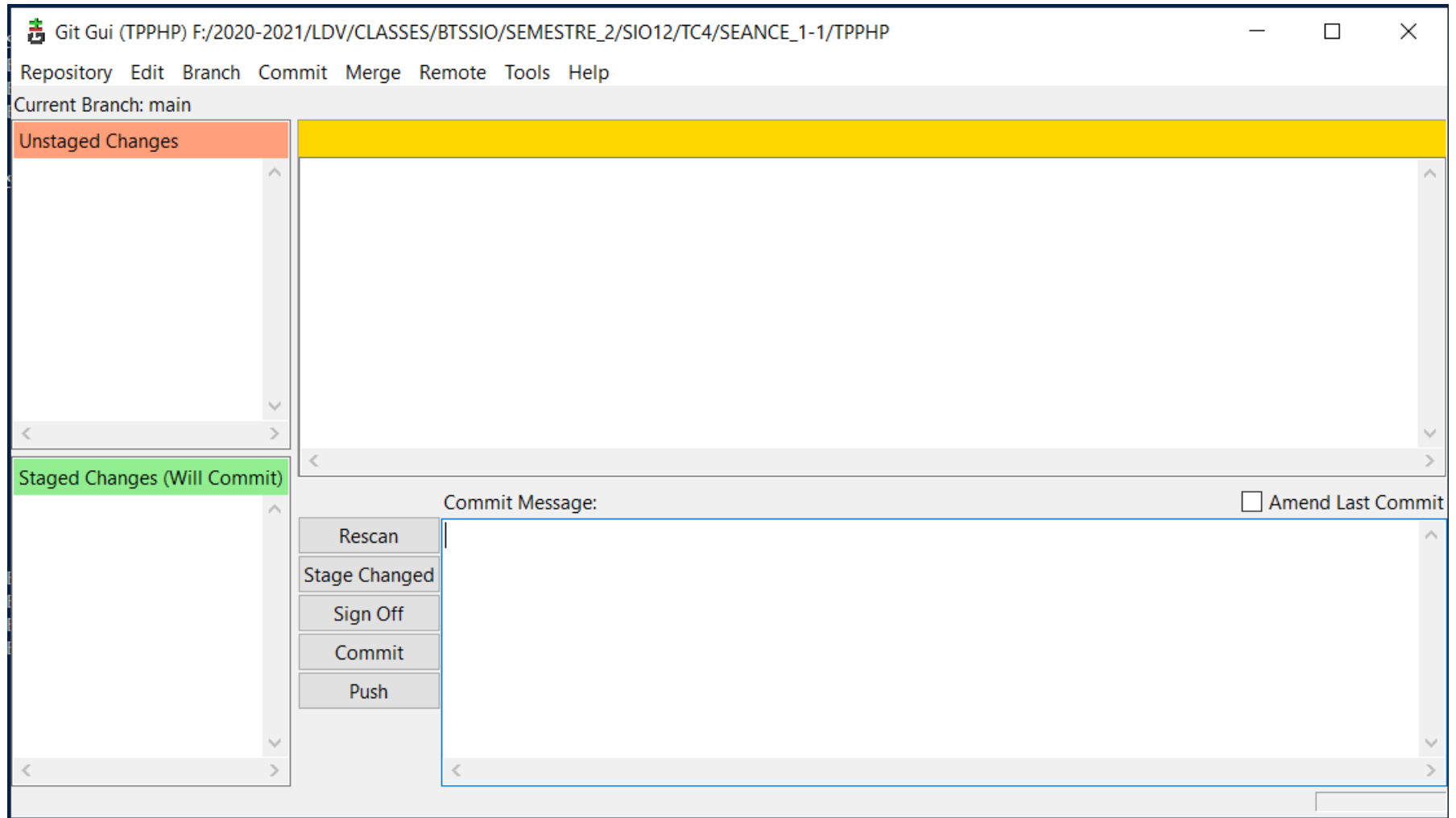
index 3b77c04..a5d2dcd 100644
@@ -1,2 +1,4 @@
apples
+pears
oranges
+bananas

9. files impacted by selected commit

Comments
fruits.txt



Outil : git gui





La collaboration



- La collaboration est une tâche fondamentale dans le versioning en général et dans Github en particulier.
- Elle consiste à faire travailler plusieurs personnes sur un même projet, en traçant les différentes interventions sur le projet.

 creationSpec.php	Modifier tableau4	last week
 exercices.zip	Modifier tableau4	last week
 formulaire.php	Modif fichier	3 days ago
 tabapp22.php	Ajout des fchiers	2 years ago
 tableau1.php	test111	last year
 tableau2.php	Modif Tableau2	2 years ago

No packages published
[Publish your first package](#)

Contributors 2



arlaroussi



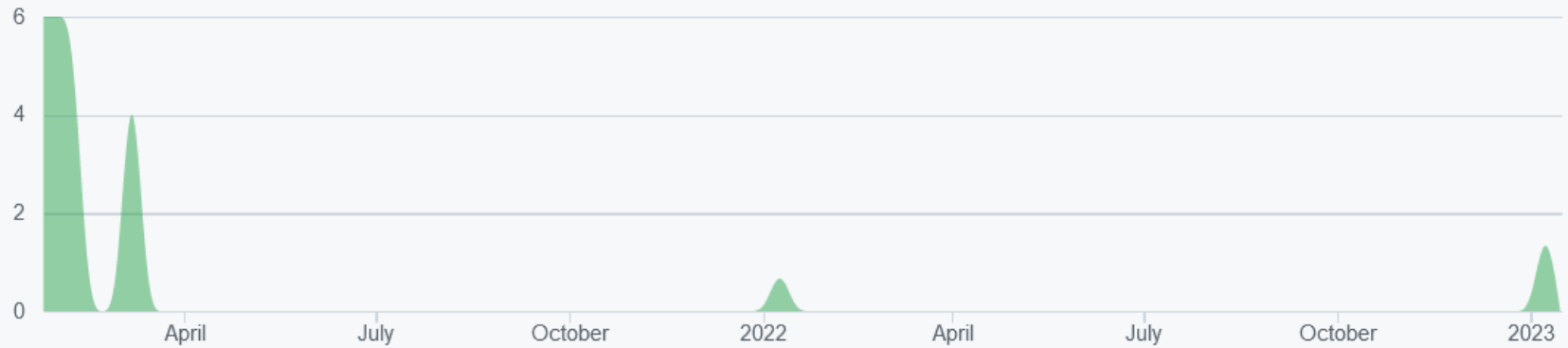
rlaroussi



Jan 24, 2021 – Jan 16, 2023

Contributions: Commits ▾

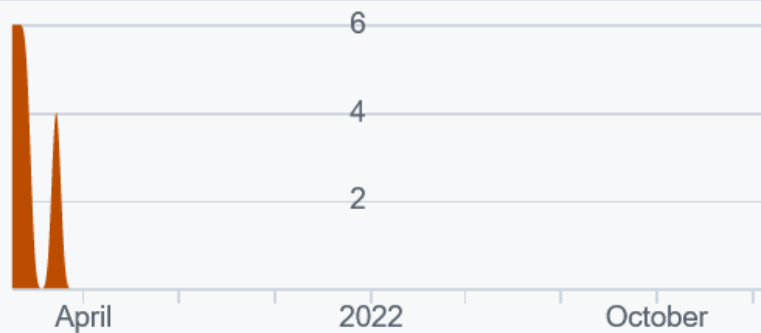
Contributions to main, excluding merge commits and bot accounts



arlaroussi

#1

24 commits 1,633 ++ 739 --



rlaroussi

#2

3 commits 7 ++ 6 --





A partir du lien Settings d'un projet, on arrive à la page suivante :

General

Access

Collaborators

Moderation options

Code and automation

Branches

Tags

Actions

Webhooks

Environments

Codespaces

Pages

Security

Code security and analysis

Deploy keys

Secrets and variables

Who has access

PUBLIC REPOSITORY

This repository is public and visible to anyone.

Manage

DIRECT ACCESS

1 has access to this repository.

0 collaborators. 1 invitation.

Manage access


Add people

Select all

Type

Find a collaborator...

☐



rlaroussi

Awaiting rlaroussi's response

Invite expired

Remove

< Previous

Next >



Add a collaborator to **PROGPHP**

🔍 Search by username, full name, or email

Select a collaborator above

Manage access

Add people

☐ Select all

Type ▾

🔍 Find a collaborator...



rlaroussi

Awaiting rlaroussi's response

Invite expired

Remove

< Previous Next >



Who has access

PUBLIC REPOSITORY



This repository is public and visible to anyone.

[Manage](#)

DIRECT ACCESS



1 has access to this repository.
[0 collaborators](#). [1 invitation](#).

Manage access

[Add people](#)

☐ Select all

Type ▾

🔍 Find a collaborator...



rlaroussi

Awaiting rlaroussi's response

Pending Invite

Remove

[< Previous](#) [Next >](#)



Danger Zone

Change repository visibility

This repository is currently public.

[Change visibility](#)

Transfer ownership

Transfer this repository to another user or to an organization where you have the ability to create repositories.

[Transfer](#)

Archive this repository

Mark this repository as archived and read-only.

[Archive this repository](#)

Delete this repository

Once you delete a repository, there is no going back. Please be certain.

[Delete this repository](#)



Les branches

Définition et intérêts des branches

- Créer une **branche** signifie diverger de la ligne principale de développement et continuer à travailler sans se préoccuper de cette ligne principale.
- La branche par défaut dans Git quand vous créez un dépôt s'appelle main (***master***) et elle pointe vers le dernier des commits réalisé.
- Créer une branche, c'est en quelque sorte comme créer une "copie" de votre projet pour développer et tester de nouvelles fonctionnalités sans impacter le projet de base.
- Une branche, dans Git, est simplement un pointeur vers un commit (une branche n'est qu'un simple fichier contenant les 40 caractères de l'empreinte SHA-1 du commit sur lequel elle pointe).

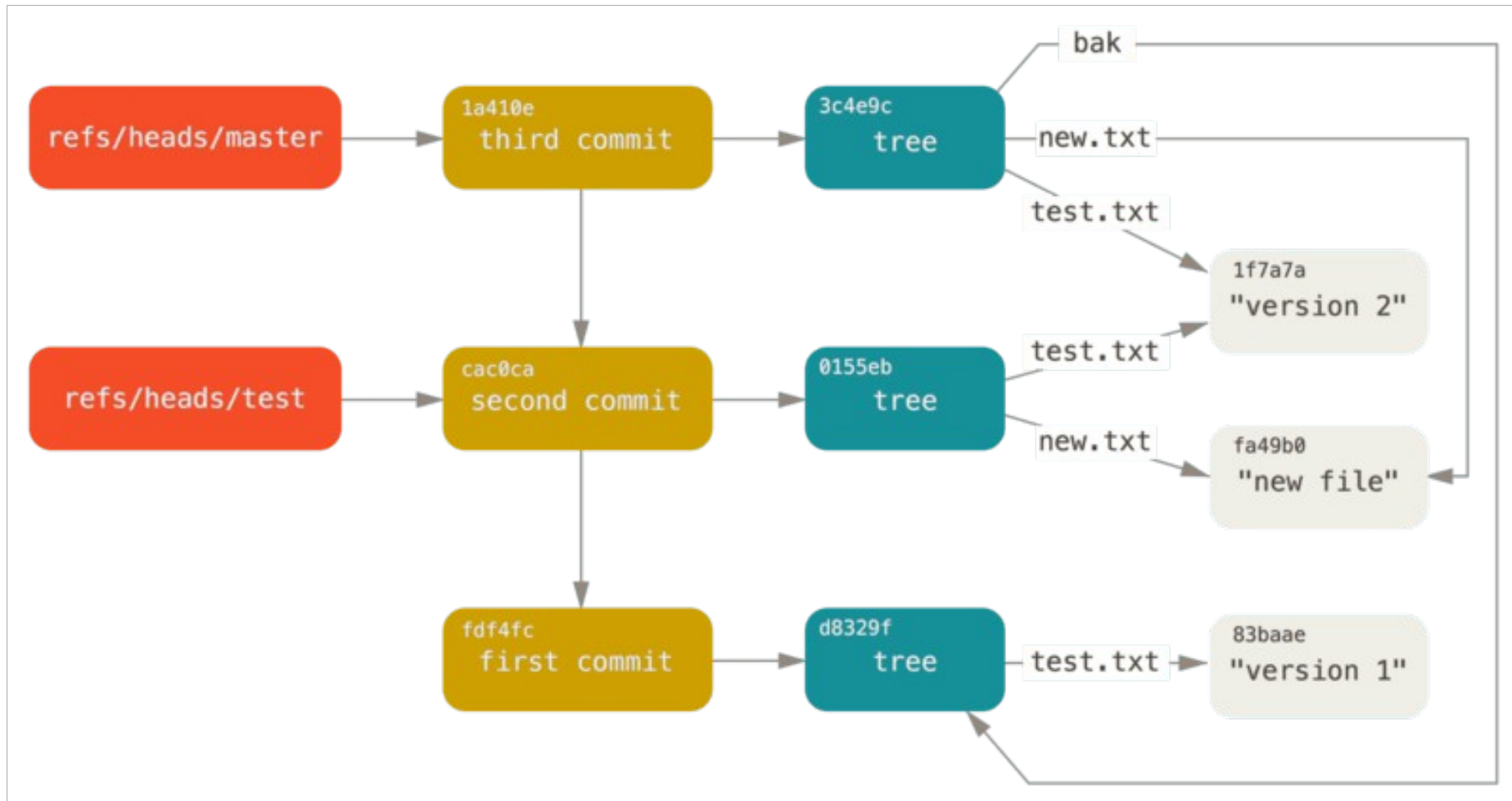
Pourquoi des branches ?

Pouvoir se lancer dans des évolutions ambitieuses en ayant toujours la capacité de revenir à une version stable que l'on peut continuer à maintenir indépendamment.

Pouvoir tester différentes implémentations d'une même fonctionnalité de manière indépendante.



Structure des branches





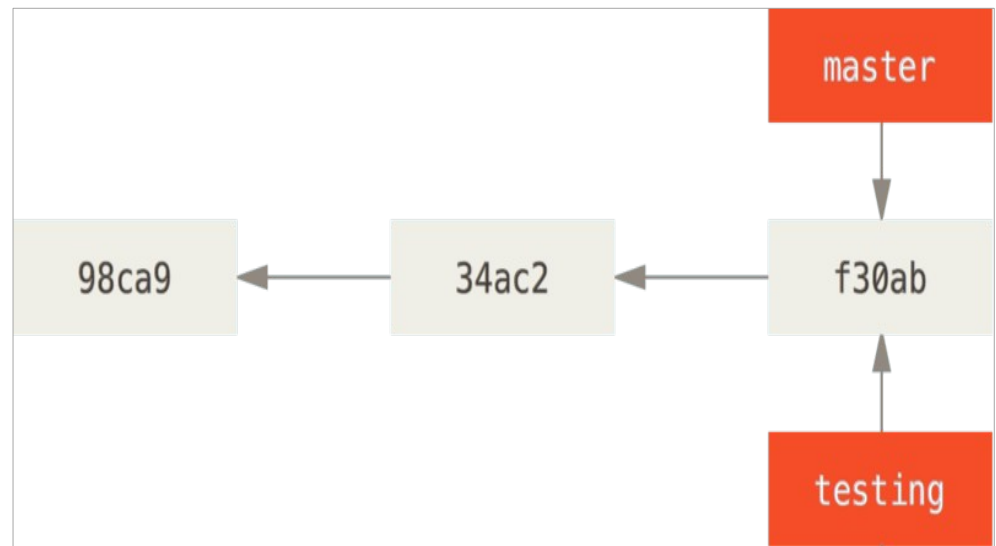
Manipulation des branches

Pour créer une nouvelle branche, on utilise la commande :

```
git branch nom_branche
```

```
git checkout -b nom_branche //Crée une branche et l'active
```

```
git branch testing
```



Pour savoir sur quelle branche vous vous trouvez, git utilise le pointeur HEAD.

Le basculement d'une branche vers l'autre passe par la commande :

```
git checkout nom_branche
```



Manipulation des branches

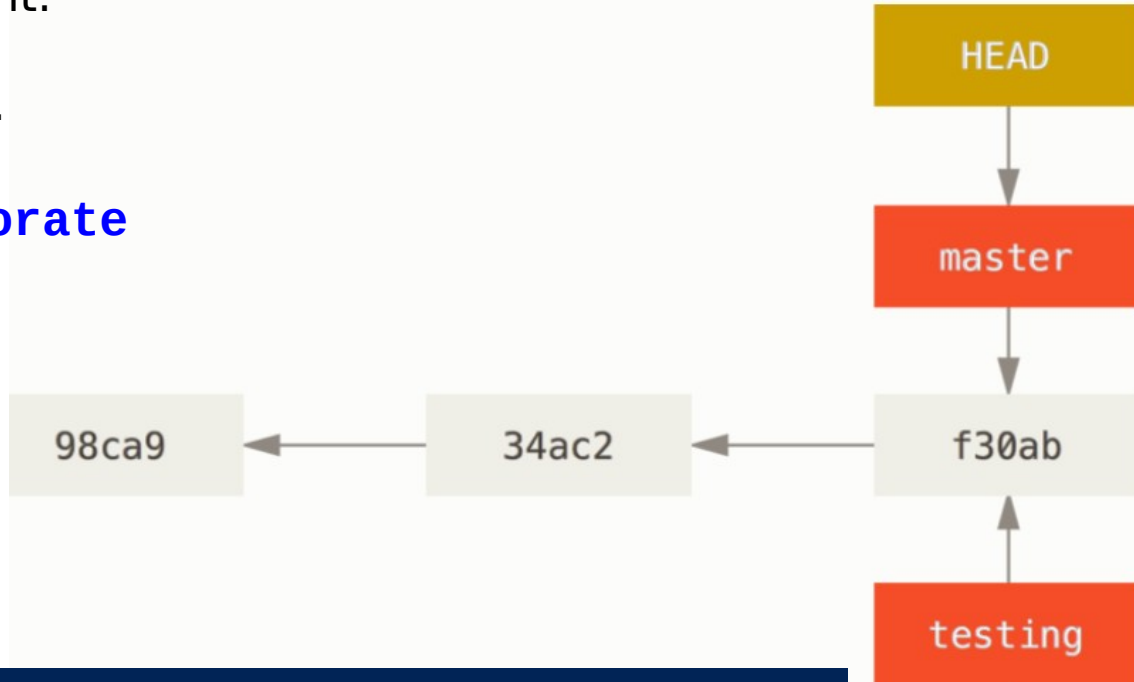
On peut vérifier cela facilement grâce à la commande git log qui vous montre vers quoi les branches pointent.

Il s'agit de l'option --decorate.

git log --oneline --decorate

git status

Vous donne la branche courante



```
>>
aaa7a7e (HEAD -> develop, origin/main, origin/develop, main) branche développement 2
a73dd25 Nouvelle branche développement
f6610e9 Modification fichiers P00
9f56e49 Ajout dossier P00
71308a7 Ajout Partiel
a332280 Modif Tableau2
4e5db54 Modif tableau1
b866ab9 Ajout creationSpec
0c74f06 Ajout des fichiers
2fcd9c9 First Commit
```



Manipulation des branches

Pour supprimer une branche, on utilise la commande :

```
git branch -d nom_branche
```

Fusionner une branche dans la branche courante

```
git merge nom_branche
```

Pousser des modification vers une branche

```
git push -u origin nom_branche
```

Afficher les différences entre deux branches

```
git diff nom_branche1...nom_branche2
```

Lister les branches distantes

```
git branch -r
```