Gestion de versions : Git

Emmanuel Grolleau

Observatoire de Paris – LESIA – Service d'Informatique Scientifique

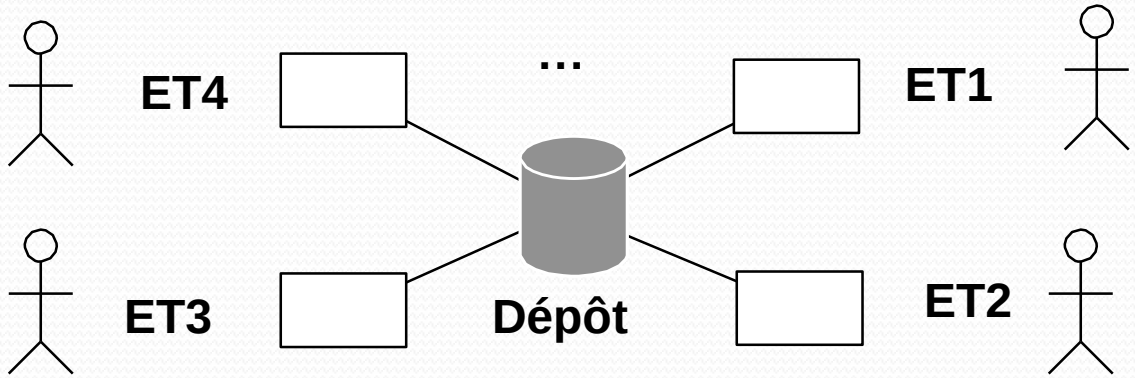
Master 2 « Outils et Systèmes de l'Astronomie et de l'Espace »

- **Pourquoi un outil de gestion de version ?**

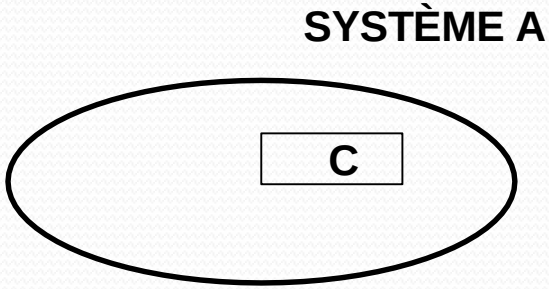
Solution : utiliser un outil de gestion de version

- Permettre à plusieurs personnes de travailler en parallèle
  - Partage intègre de fichiers
    - Gérer les accès
    - Indiquer les conflits
    - Notifier les modifications
- Gérer les versions des composants
- Garantir traçabilité source à exécutable
- Revenir à une version antérieure

Permettre à plusieurs personnes de travailler en parallèle



Problème des mises a jour simultanées

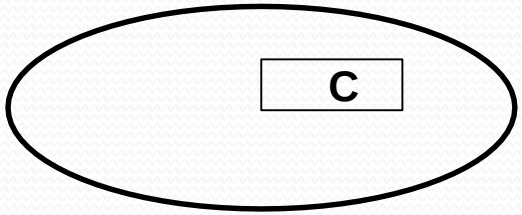


Problème des mises a jour simultanées

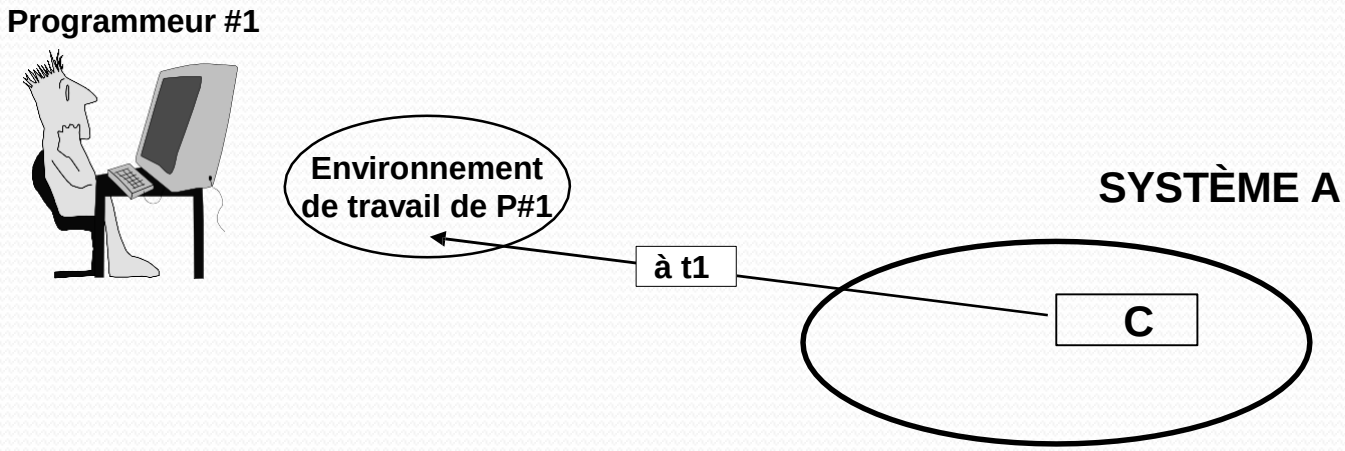
Programmeur #1

Environnement de travail de P#1

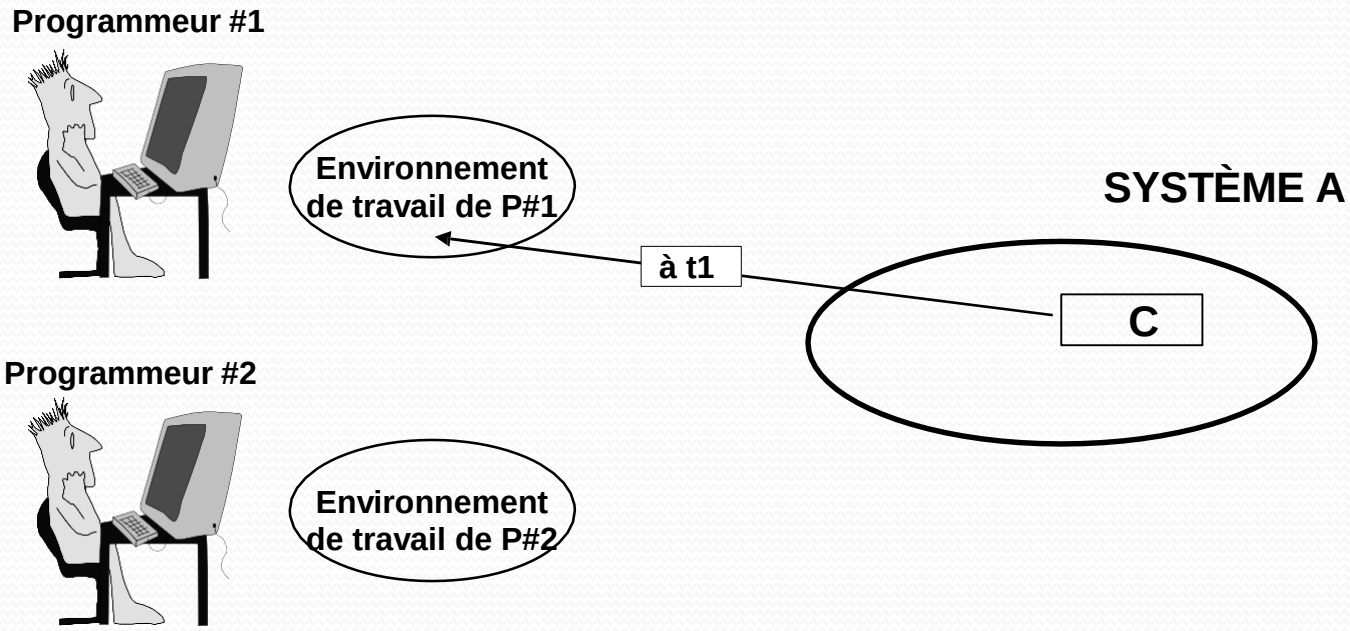
SYSTÈME A



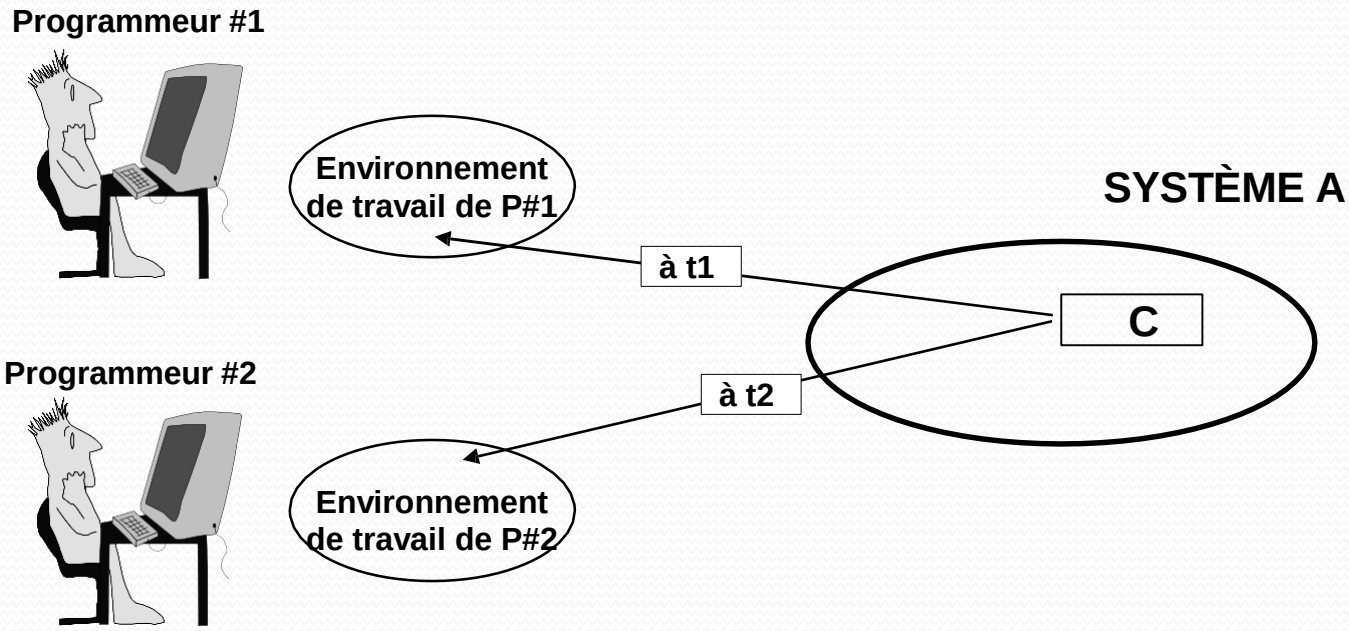
Problème des mises a jour simultanées



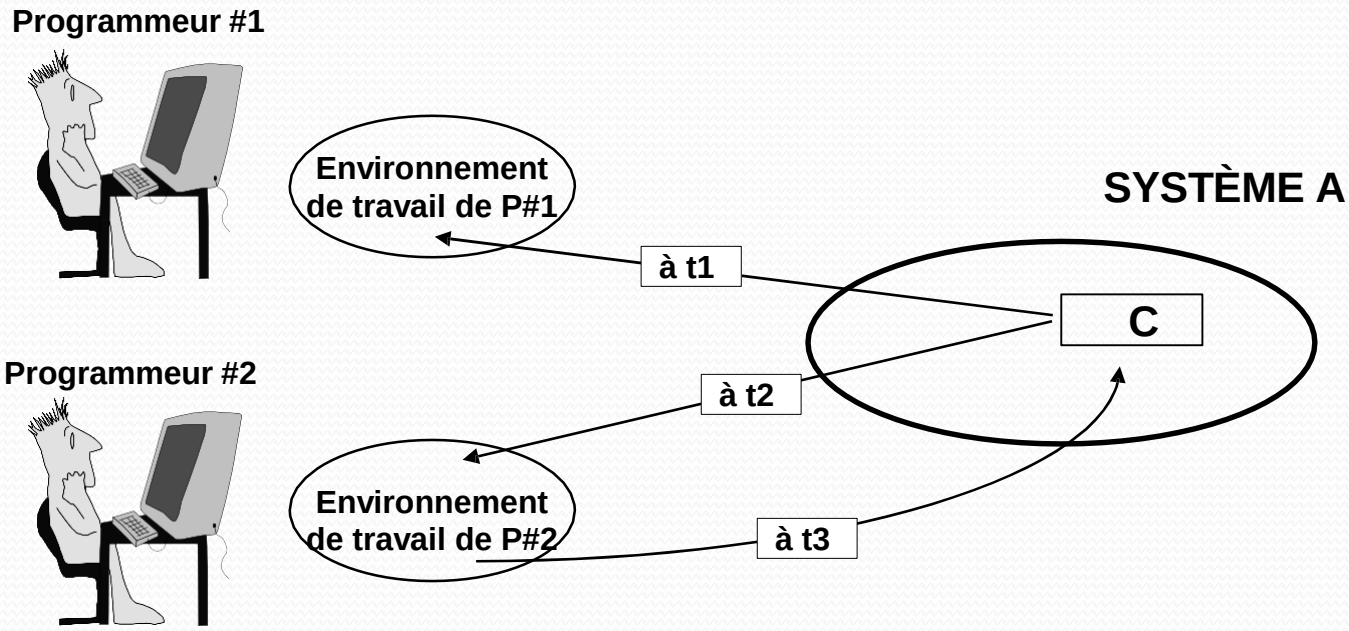
Problème des mises a jour simultanées



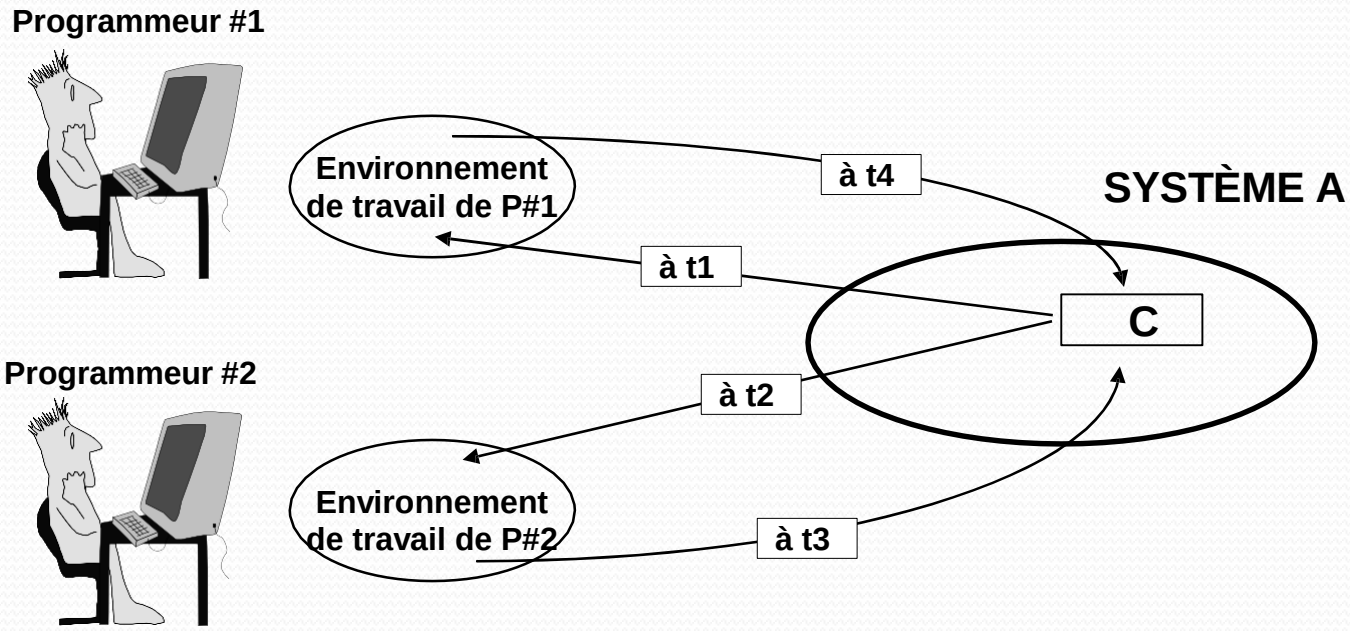
Problème des mises a jour simultanées



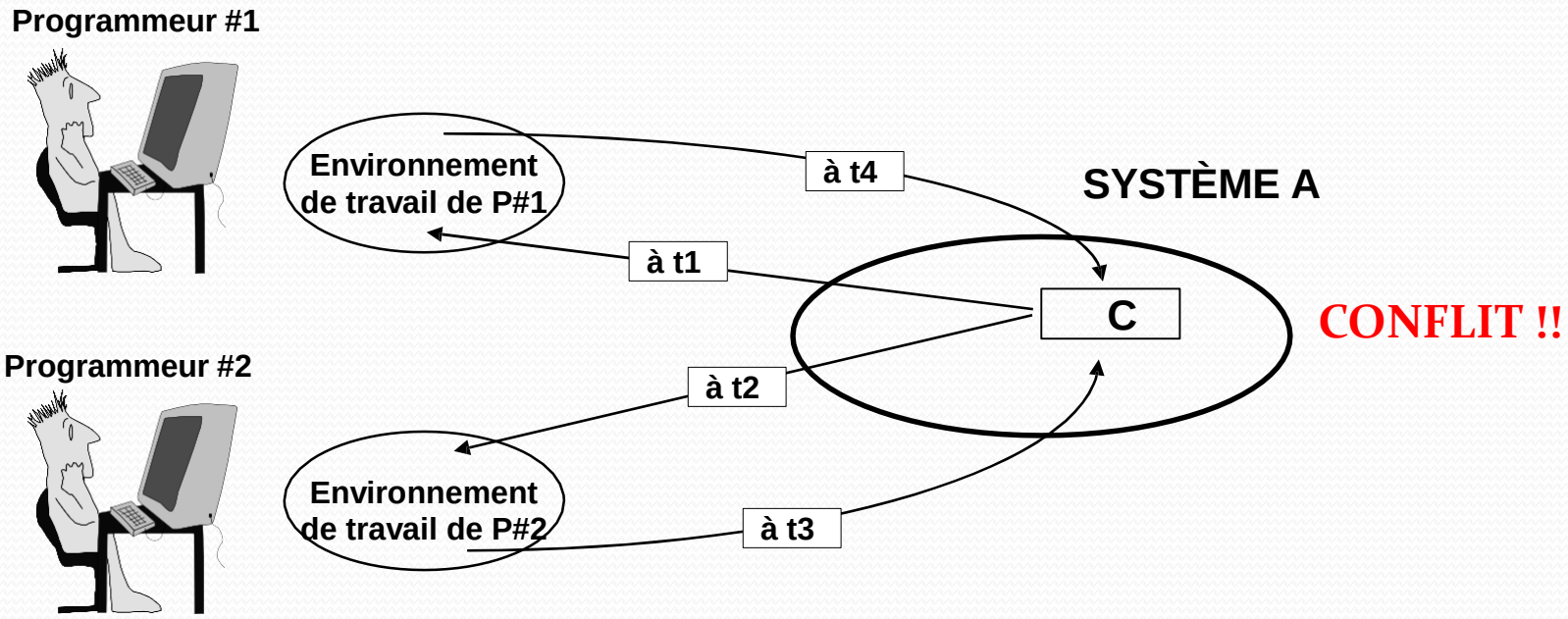
Problème des mises a jour simultanées



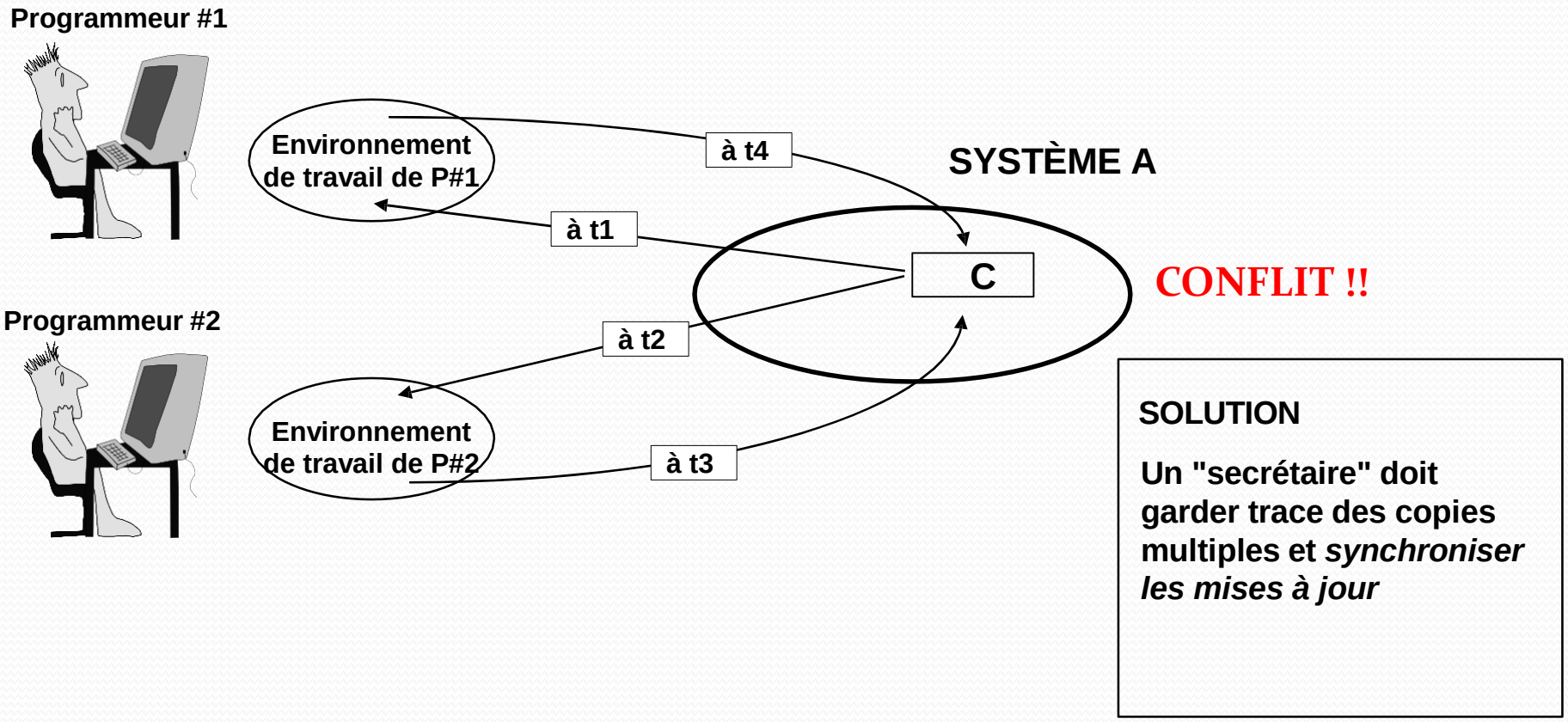
Problème des mises a jour simultanées



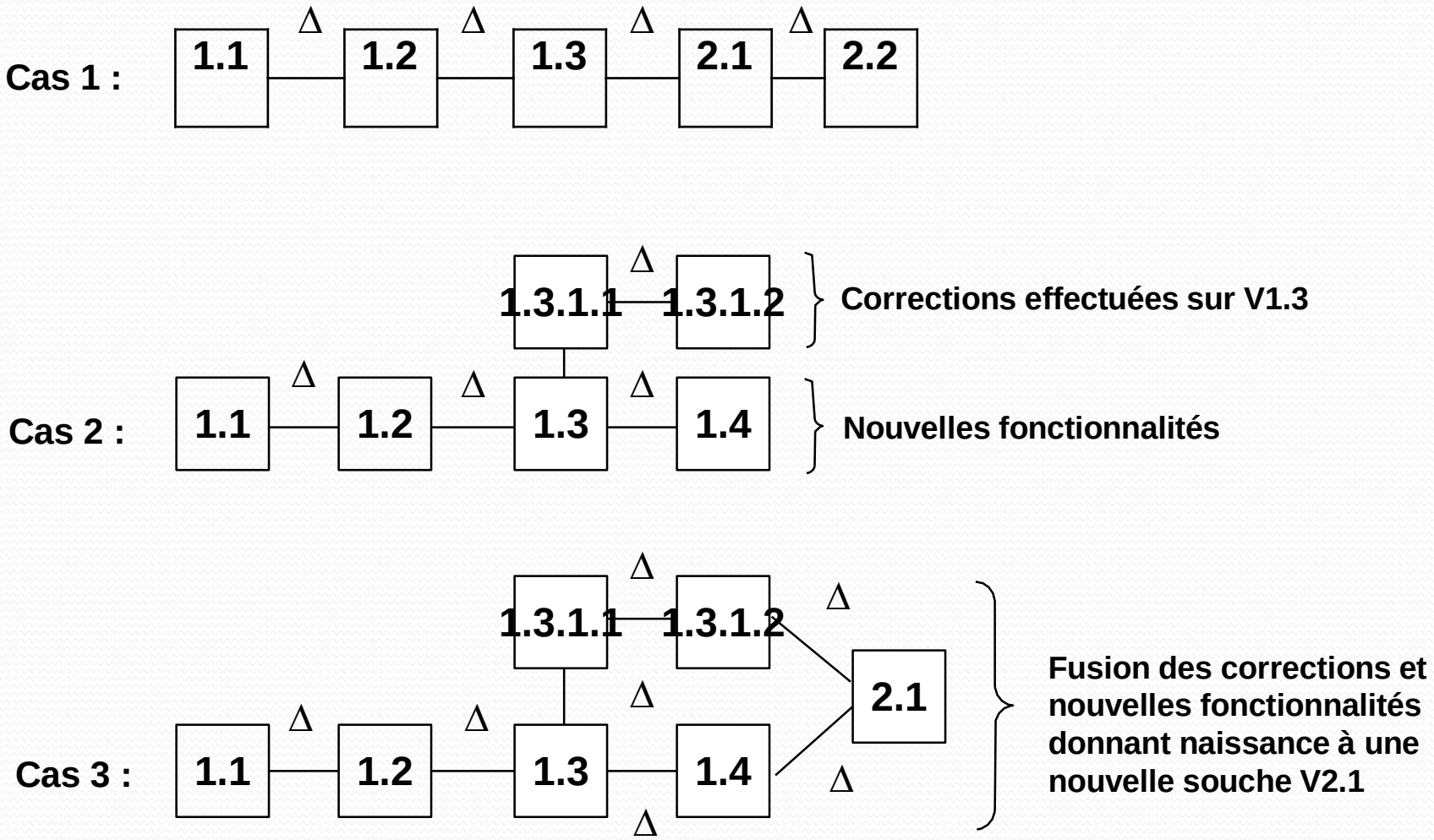
Problème des mises a jour simultanées



Problème des mises a jour simultanées



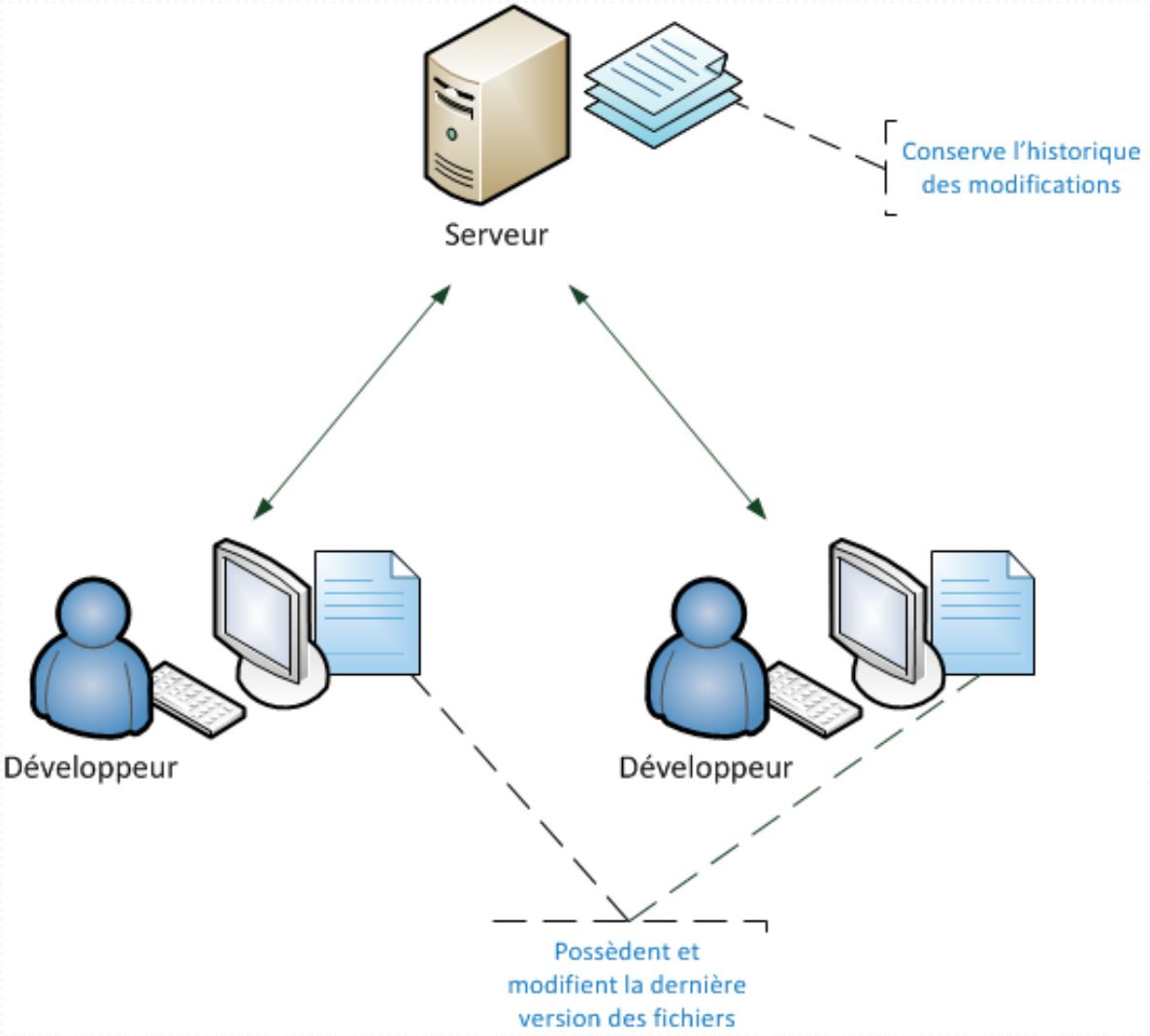
Gérer les versions des composants



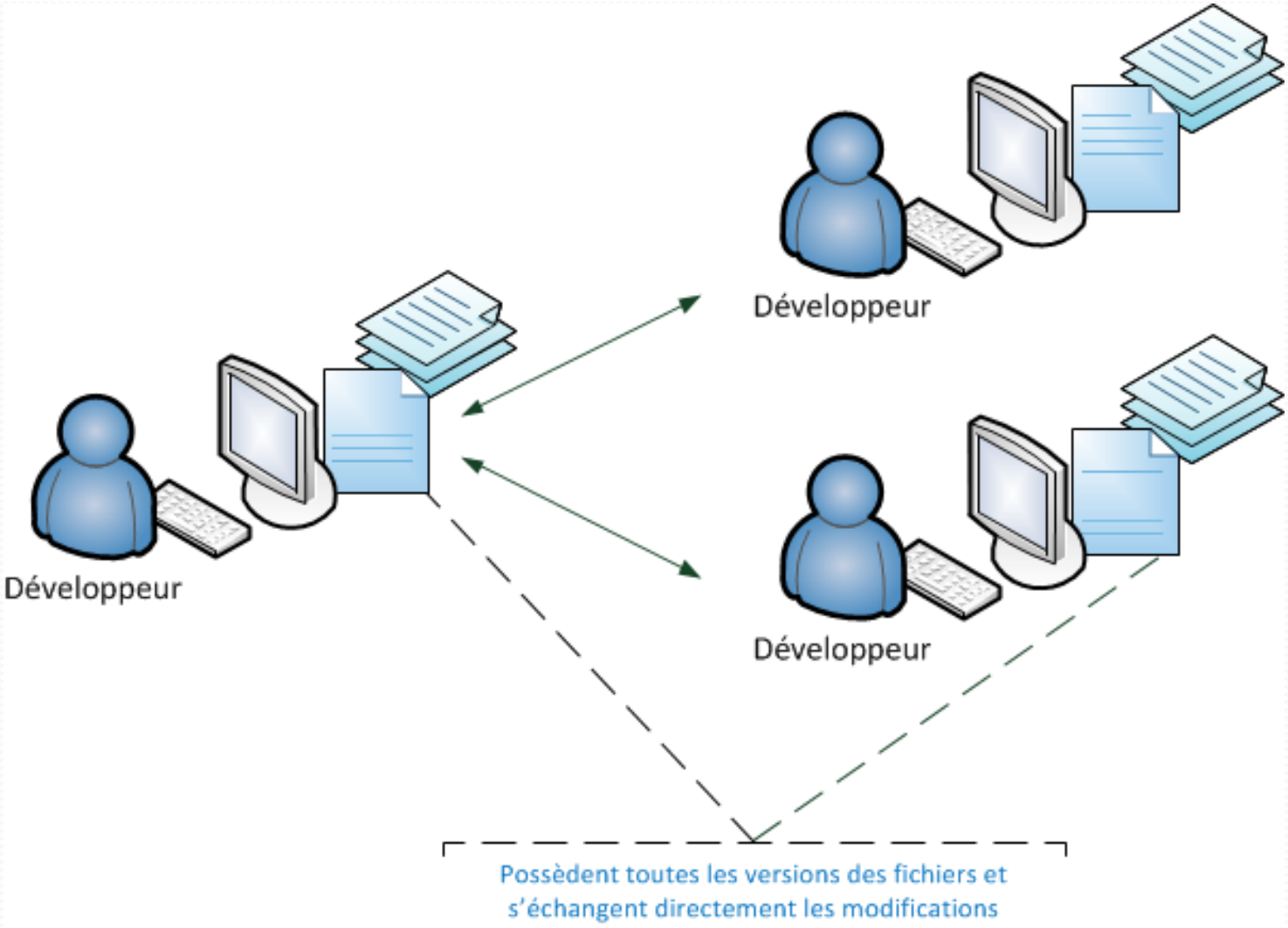
Outils centralisés et distribués

- Il existe deux types principaux de logiciels de gestion de versions
- Les logiciels centralisés : un serveur conserve les anciennes versions des fichiers et les développeurs s'y connectent pour prendre connaissance des fichiers qui ont été modifiés par d'autres personnes et pour y envoyer leurs modifications.
  - Subversion, CVS
- Les logiciels distribués : il n'y a pas de serveur, chacun possède l'historique de l'évolution de chacun des fichiers. Les développeurs se transmettent directement entre eux les modifications, à la façon du peer-to-peer.
  - Git

Outil de gestion de version **centralisé** (CVS, SVN)



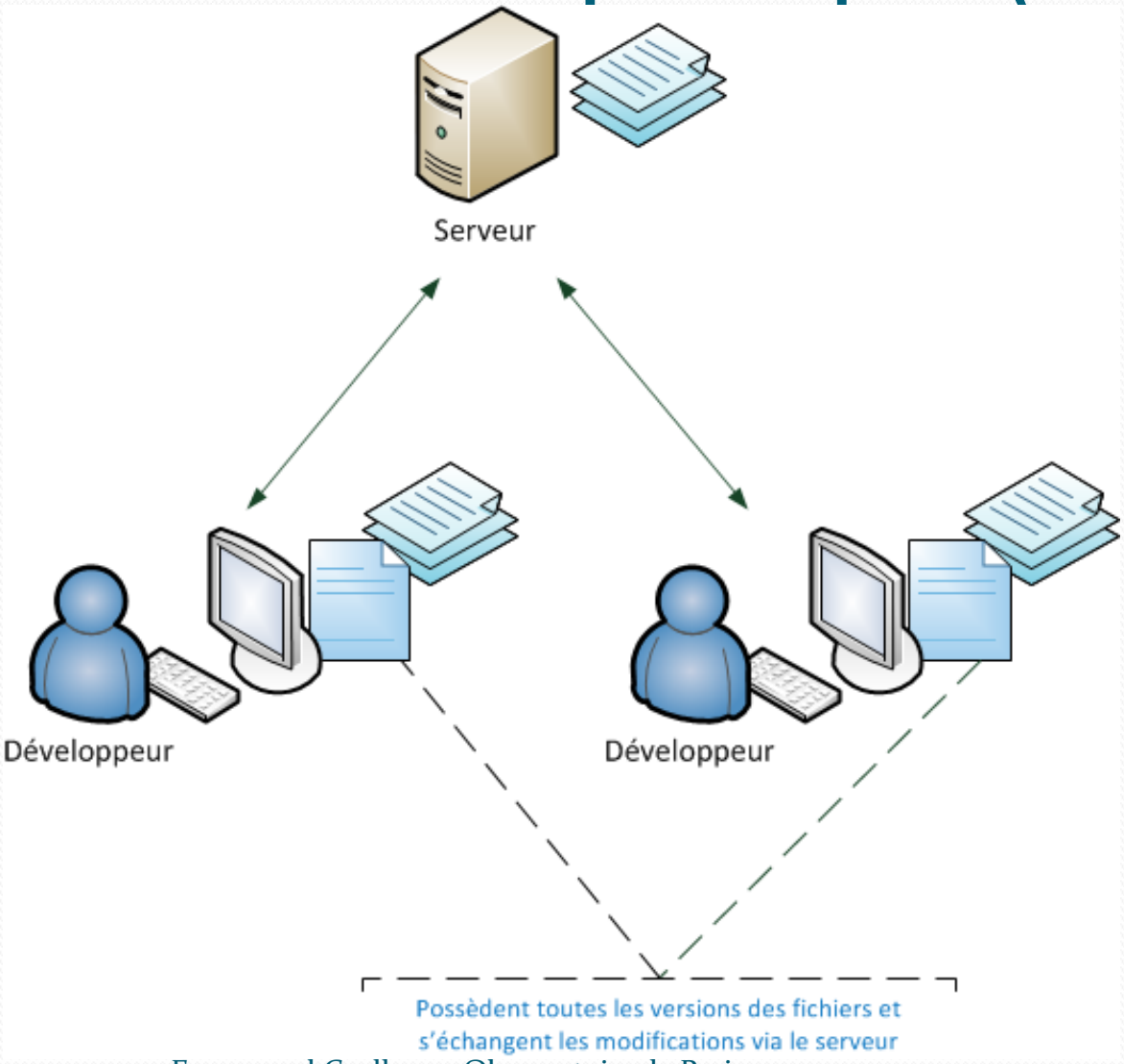
Outil de gestion de version **distribué**



Outil de gestion de version distribué : dans la pratique

- Dans la pratique, les logiciels distribués sont rarement utilisés comme sur le schéma précédent.
- Même lorsque les logiciels sont capables de fonctionner en mode distribué, on utilise très souvent un serveur qui sert de point de rencontre entre les développeurs.
- Le serveur connaît l'historique des modifications et permet l'échange d'informations entre les développeurs, qui eux possèdent également l'historique des modifications.
- Avantage : meilleure gestion des branches

Outil de gestion de version distribué : dans la pratique (Git)



Un outil de gestion de version : Git

Git est un logiciel de gestion de sources et contrôle de versions.

Il est principalement utilisé pour maintenir le code source ou la documentation.

Ses principales fonctionnalités sont :

- garder un historique des différentes versions des fichiers d'un projet
- permettre le retour à une version antérieure quelconque
- garder un historique des modifications
- permettre un accès à ces fichiers, en local ou via un réseau
- permettre à des utilisateurs distincts de travailler ensemble sur les mêmes fichiers

Notions générales

- Dépôt local (local repository)
  - Répertoire situé en local sur le poste utilisateur et qui contient les données relatives aux projets.
- Dépôt distant (remote repository)
  - Répertoire situé à distance sur un serveur et qui contient les données relatives aux projets.
  - On accède au dépôt distant via une URL distante Ex : https://nomserveur/git/repo1
  - Service en ligne fourni par **Github**, **Gitlab, Bitbucket**
- Projets
  - Répertoire contenant les fichiers et dossiers du projet. Un dépôt Git peut contenir une infinité de projets.
- Révision
  - Modification faite au dépôt
  - Indicateur s'incrémentant à chaque opération et permettant de revenir à une version donnée d'un ou plusieurs fichiers

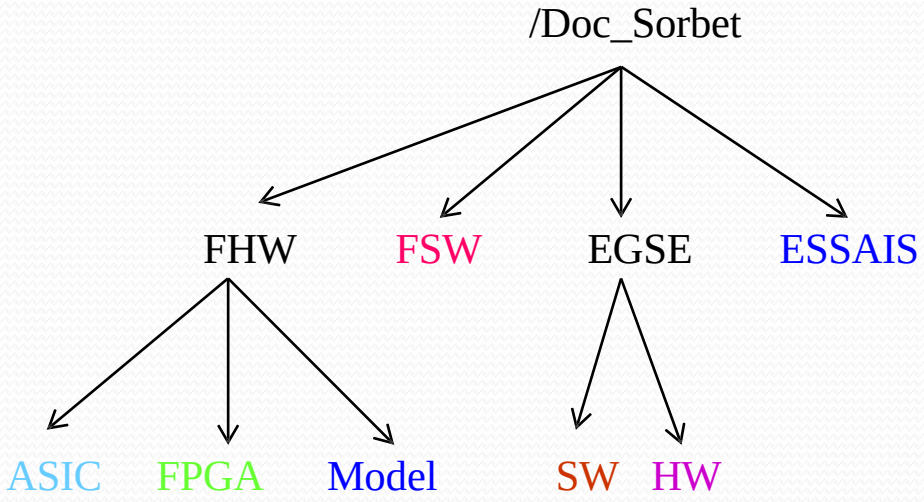
Le principe

- Dépôt contenant l'ensemble des versions
  - Local et/ou serveur
  - Garantissant l'intégrité
    - Ne pouvant être accédé que par les commandes de l'outil
  - Sécurité de gestion des accès
  - Contenant l'ensemble des versions successives

Exemple de dépôt (repository)

Le Dépôt de la documentation du projet SORBET-BepiColombo (Mercure).

/Doc_Sorbet (par défaut tous les acteurs du projet ont le droit en lecture)



Les accès en écriture :

Dekkali :

All :

Davy :

Astier :

Grolleau :

Boughedada :

Opérations de base

- **init** :
  - Permet d'initialiser localement un nouveau projet Git
  - Opération à ne réaliser qu'une seule fois en local

  cd mon_projet; git init

- **clone** <URL> :
  - Cloner en local un dépôt distant existant à partir de l'URL fournie ;
  - Opération à ne réaliser qu'une seule fois;
  - Génération d'un dépôt local de travail lié avec le dépôt distant

  git clone https://gitlab.obspm.fr/osae/2025.git

Opérations de base avec le dépôt local

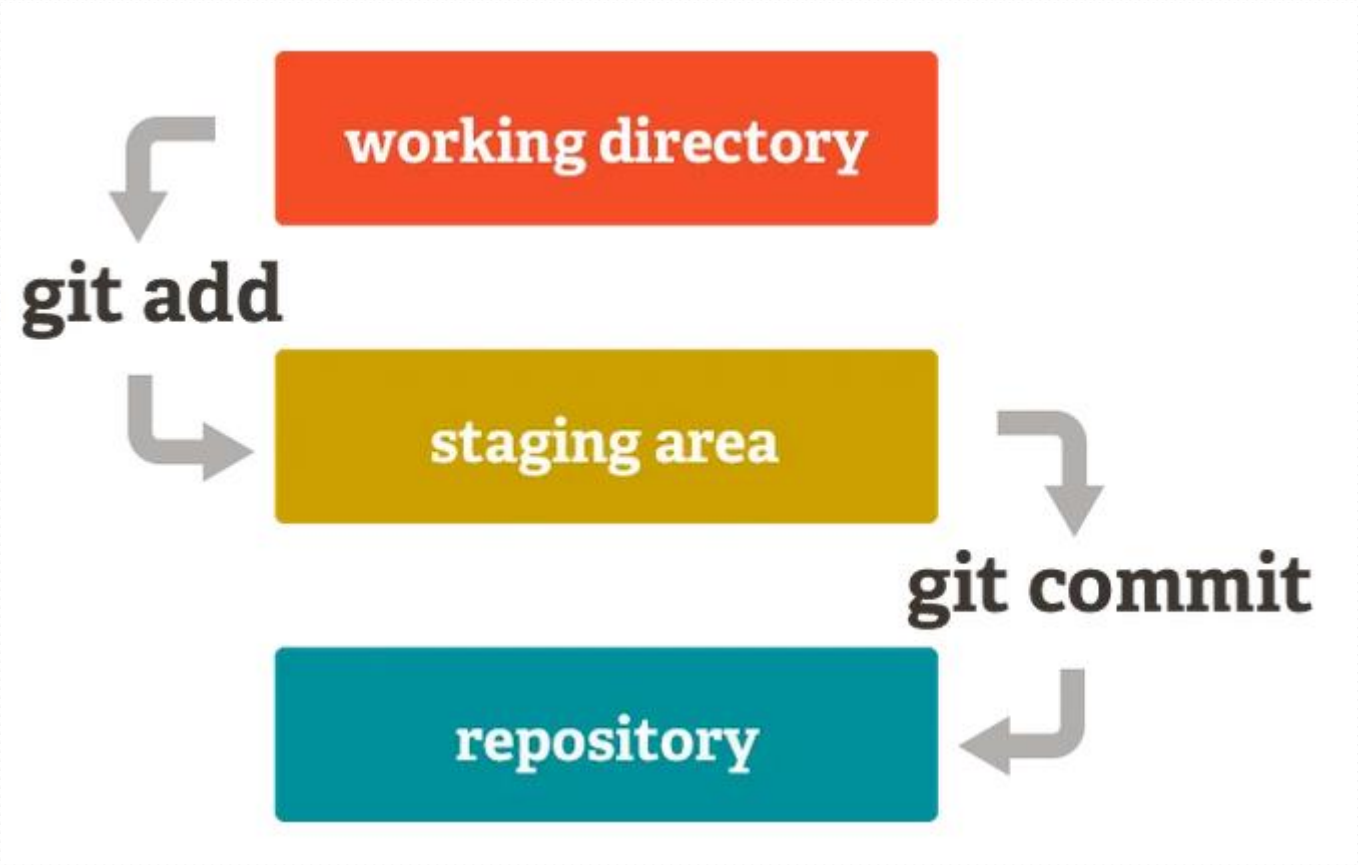
- **add** :
  - Permet d'ajouter des fichiers de notre répertoire de travail (working directory) à l'index (staging area) pour le prochain commit

  git add <file>

- **commit** :
  - Envoyer nos fichiers listés dans l'index vers notre dépôt local en indiquant un message précisant la raison des modifications

  git commit –m « Raison des modifications »

Opérations de base avec le dépôt local



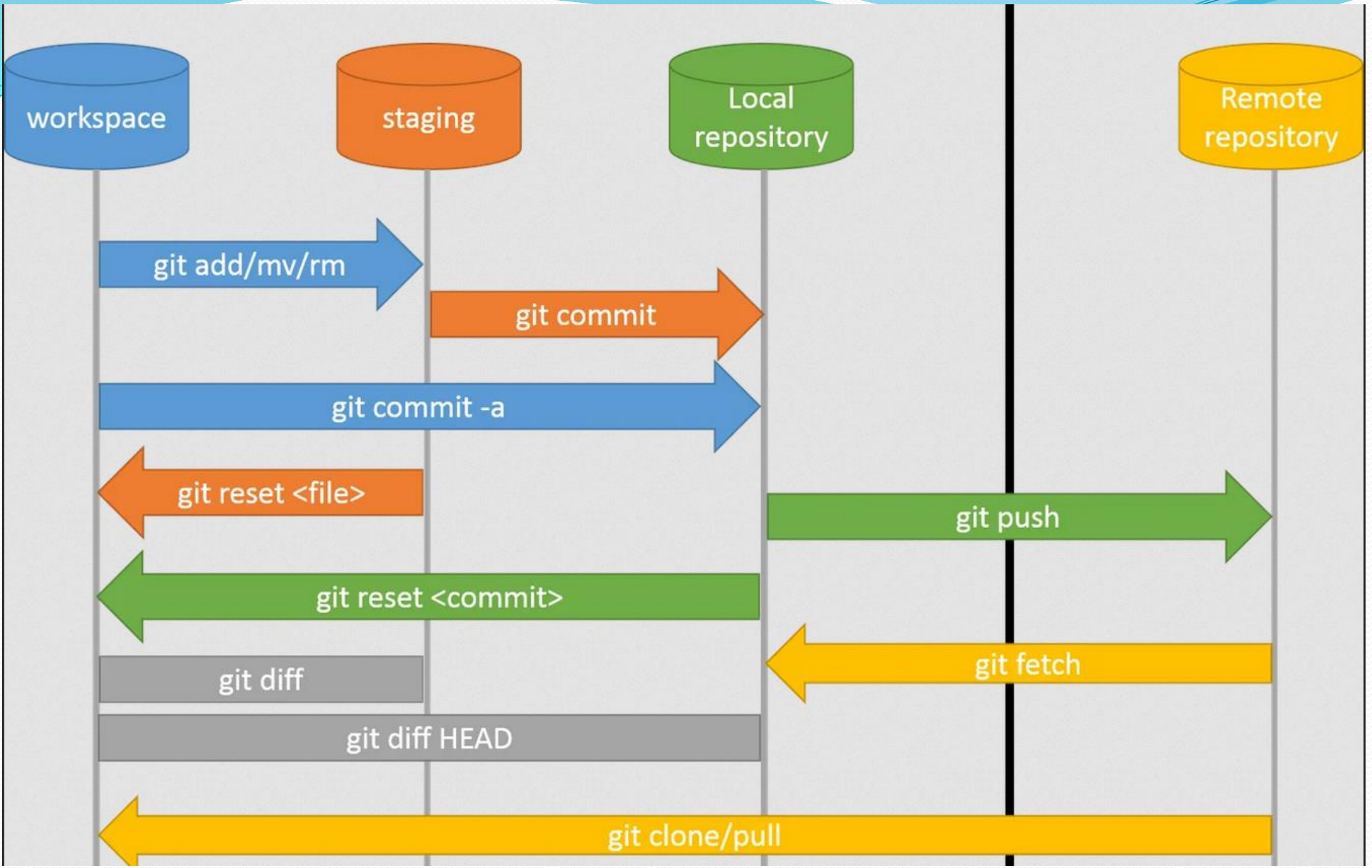
Opérations de base avec le dépôt distant

- **pull** :
  - Permet de récupérer les dernières modifications du dépôt distant;

  git pull

- **push** :
  - Permet d'envoyer nos modifications vers le dépôt distant;

  git push


workspace
staging
Local repository
Remote repository
git add/mv/rm
git commit
git commit -a
git reset <file>
git push
git reset <commit>
git fetch
git diff
git diff HEAD
git clone/pull

Scénario nominal

1) Cloner le projet à partir du dépôt

git clone https://gitlab.obspm.fr/osae/2025.git

2) Dans le répertoire du groupe, créer, supprimer ou renommer des fichiers ou des repertoires et les ajouter à l'index.

cd 2025/Groupes/Groupe01, git add file | git rm file | git mv file1 file2

3) Commit de vos changements vers le dépôt local.

git commit -m " message sur le commit "

4) Envoyer les modifications vers le dépôt distant

git push

5) Mettre à jour le repertoire de travail depuis le dépôt distant (récupérer les modifications des autres développeurs)

git pull

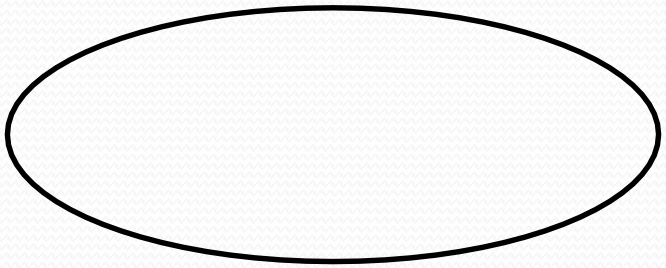
Une commande indispensable : *git help <nom de la commande>*

Scénario nominal

Programmeur #1

Dépôt local de P#1

Chef de projet (CP)



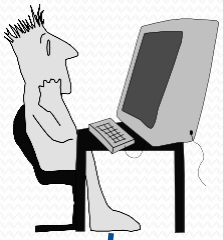
Dêpôt distant Gitlab (Repository)

Scénario nominal

Programmeur #1

Dépôt local de P#1

Chef de projet (CP)



1. Création d'un projet en ligne

Fichier1

Fichier2

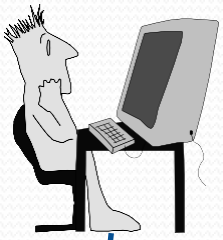
Dêpôt distant Gitlab (Repository)

Scénario nominal

Programmeur #1

Dépôt local de P#1

Chef de projet (CP)



1. Création d'un projet en ligne

2. clone

Fichier1

Fichier2

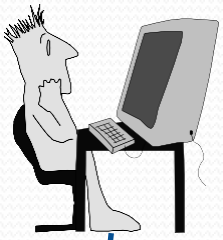
Dêpôt distant Gitlab (Repository)

Scénario nominal

Programmeur #1

Dépôt local de P#1

Chef de projet (CP)



Dépôt local de CP

Fichier1

Fichier2

1. Création d'un projet en ligne

2. clone

Fichier1

Fichier2

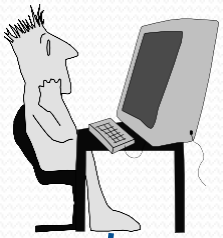
Dêpôt distant Gitlab (Repository)

Scénario nominal

Programmeur #1

Dépôt local de P#1

Chef de projet (CP)



Dépôt local de CP

Fichier1

Fichier2

3. clone

1. Création d'un projet en ligne

2. clone

Fichier1

Fichier2

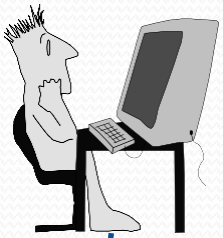
Dêpôt distant Gitlab (Repository)

Scénario nominal

Programmeur #1

Dépôt local de P#1

Chef de projet (CP)



Dépôt local de CP

Fichier1

Fichier2

3. clone

Fichier1

Fichier2

1. Création d'un projet en ligne

2. clone

Fichier1

Fichier2

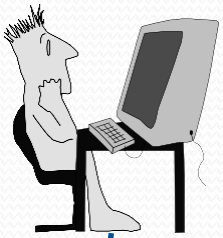
Dêpôt distant Gitlab (Repository)

Scénario nominal

Programmeur #1

Dépôt local de P#1

Chef de projet (CP)



Dépôt local de CP

Fichier1

Fichier2

3. clone

Fichier1

Fichier2

1. Création d'un projet en ligne

4. Modifications

2. clone

Fichier1

Fichier2

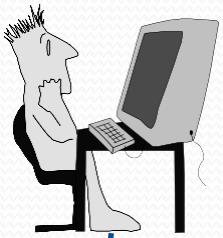
Dêpôt distant Gitlab (Repository)

Scénario nominal

Programmeur #1

Dépôt local de P#1

Chef de projet (CP)



Dépôt local de CP

Fichier1

Fichier2

3. clone

Fichier1

Fichier2

5. add/commit/ push

1. Création d'un projet en ligne

4. Modifications

2. clone

Fichier1

Fichier2

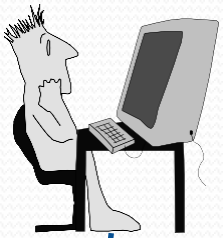
Dêpôt distant Gitlab (Repository)

Scénario nominal

Programmeur #1

Dépôt local de P#1

Chef de projet (CP)



Dépôt local de CP

Fichier1

Fichier2

3. clone

Fichier1

Fichier2

5. add/commit/ push

1. Création d'un projet en ligne

4. Modifications

2. clone

Fichier1

Fichier2

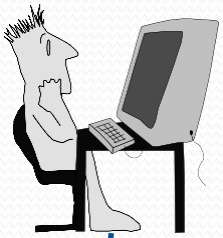
Dêpôt distant Gitlab (Repository)

Scénario nominal

Programmeur #1

Dépôt local de P#1

Chef de projet (CP)



Dépôt local de CP

Fichier1

Fichier2

3. clone

Fichier1

Fichier2

5. add/commit/ push

1. Création d'un projet en ligne

4. Modifications

2. clone

Fichier1

Fichier2

6. add/commit/ push

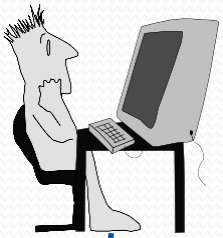
Dêpôt distant Gitlab (Repository)

Scénario nominal

Programmeur #1

Dépôt local de P#1

Chef de projet (CP)



Dépôt local de CP

Fichier1

Fichier2

3. clone

Fichier1

Fichier2

5. add/commit/ push

1. Création d'un projet en ligne

4. Modifications

7. pull

Fichier1

Fichier2

2. clone

6. add/commit/ push

Dêpôt distant Gitlab (Repository)

Les conflits

Dépôt local de P#1

Programmeur #1

Programmeur #2

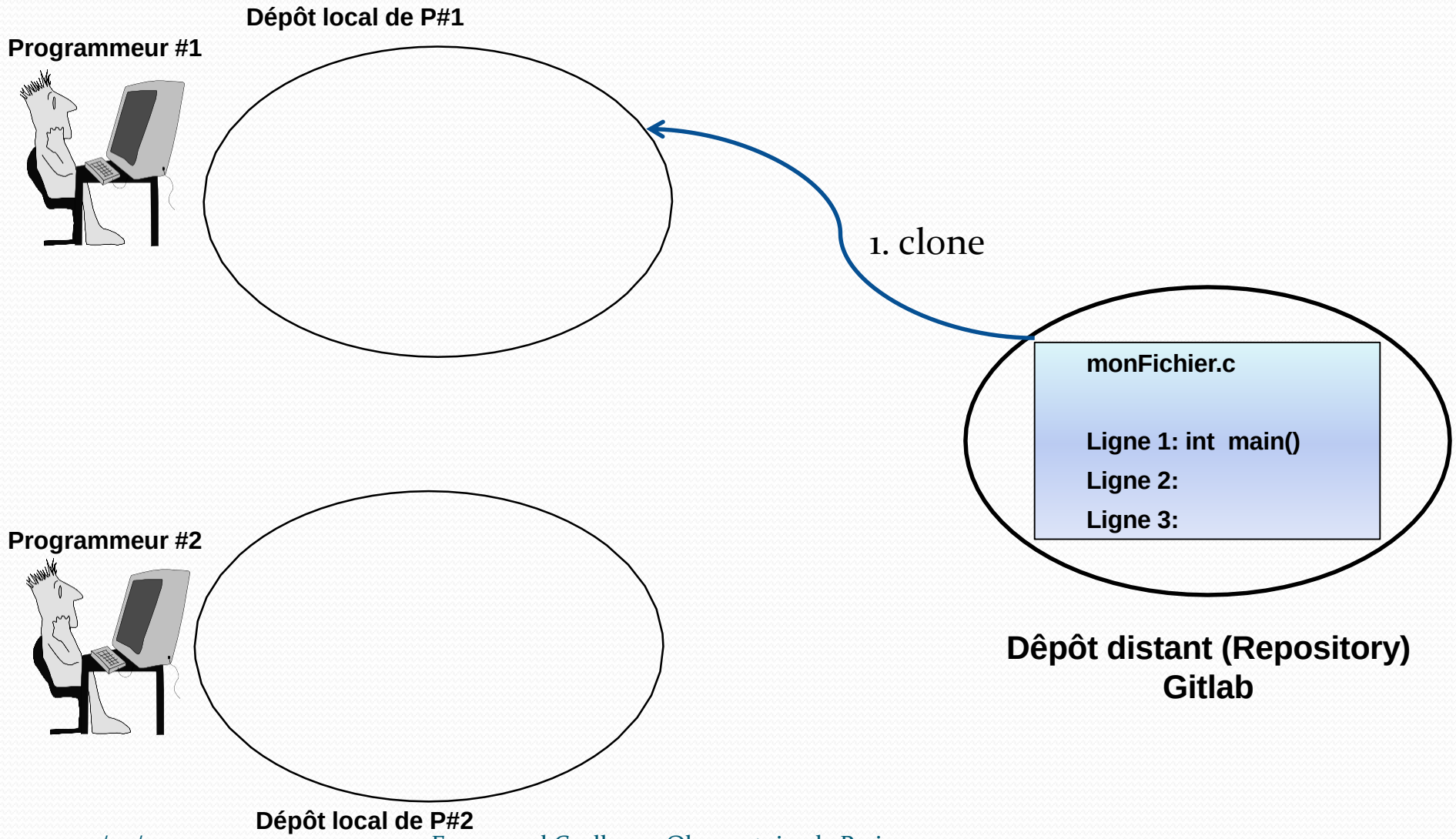
Dépôt local de P#2

monFichier.c

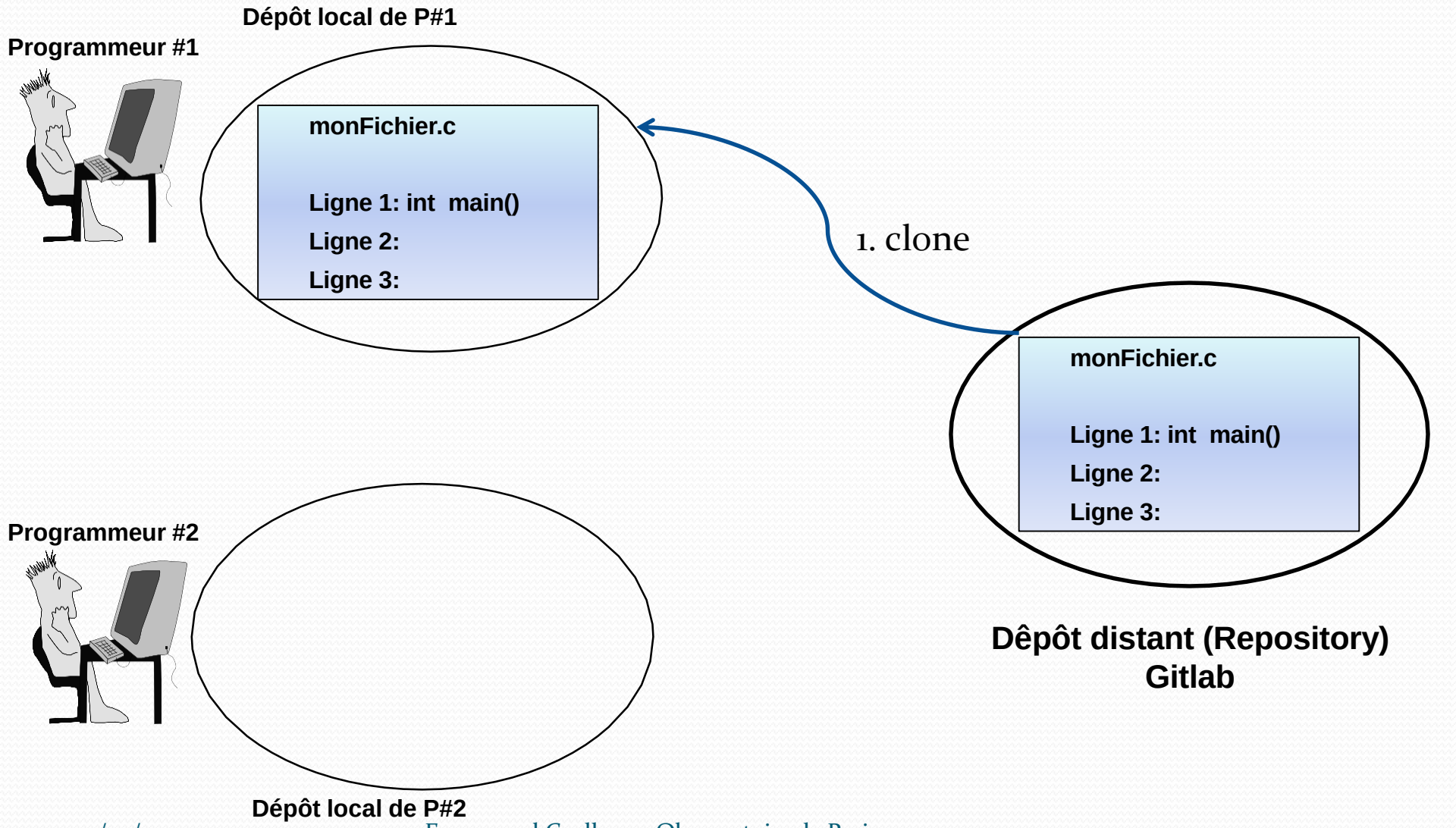
Ligne 1: int main()
Ligne 2:
Ligne 3:

Dêpôt distant (Repository)
Gitlab

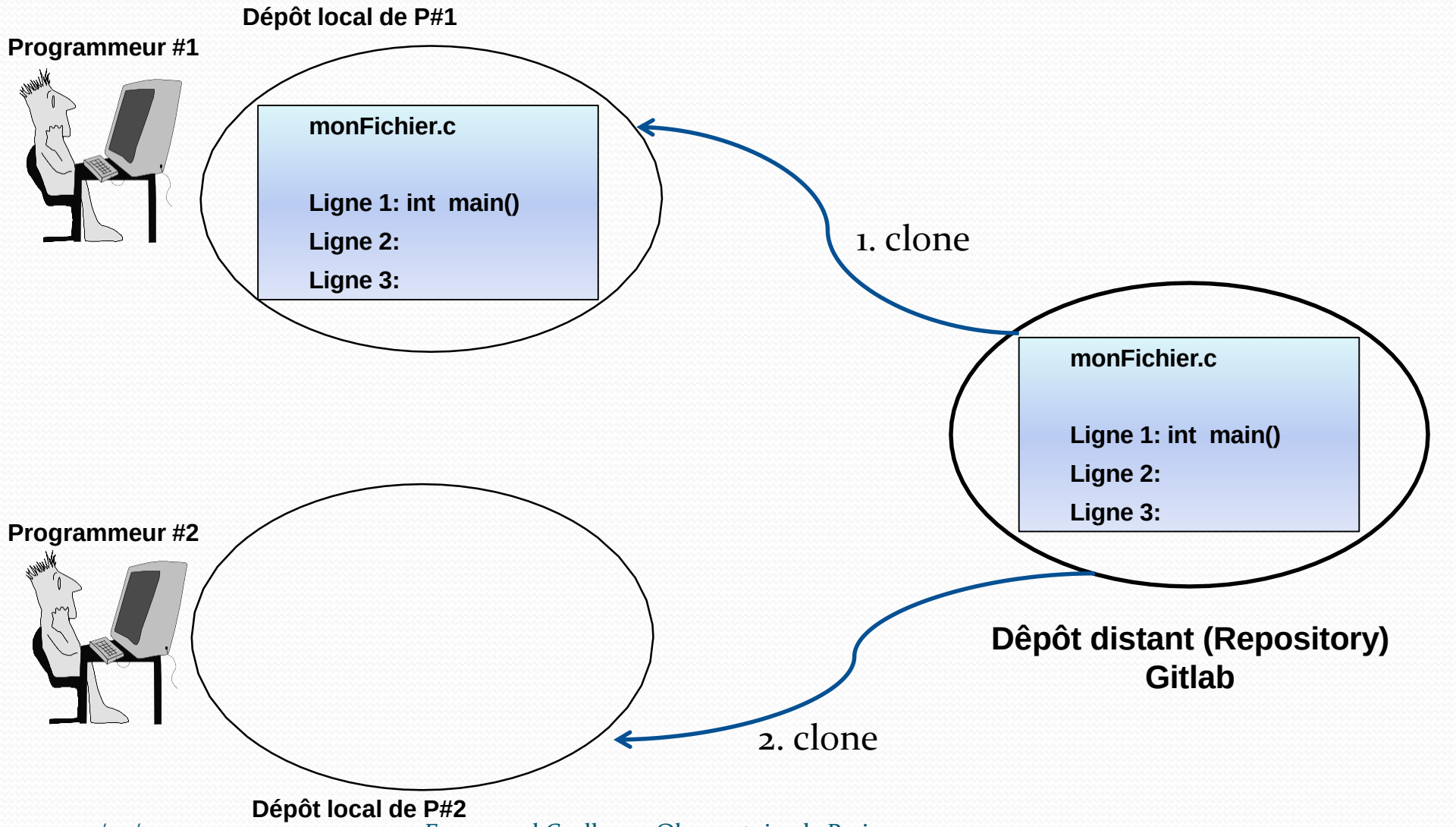
Les conflits



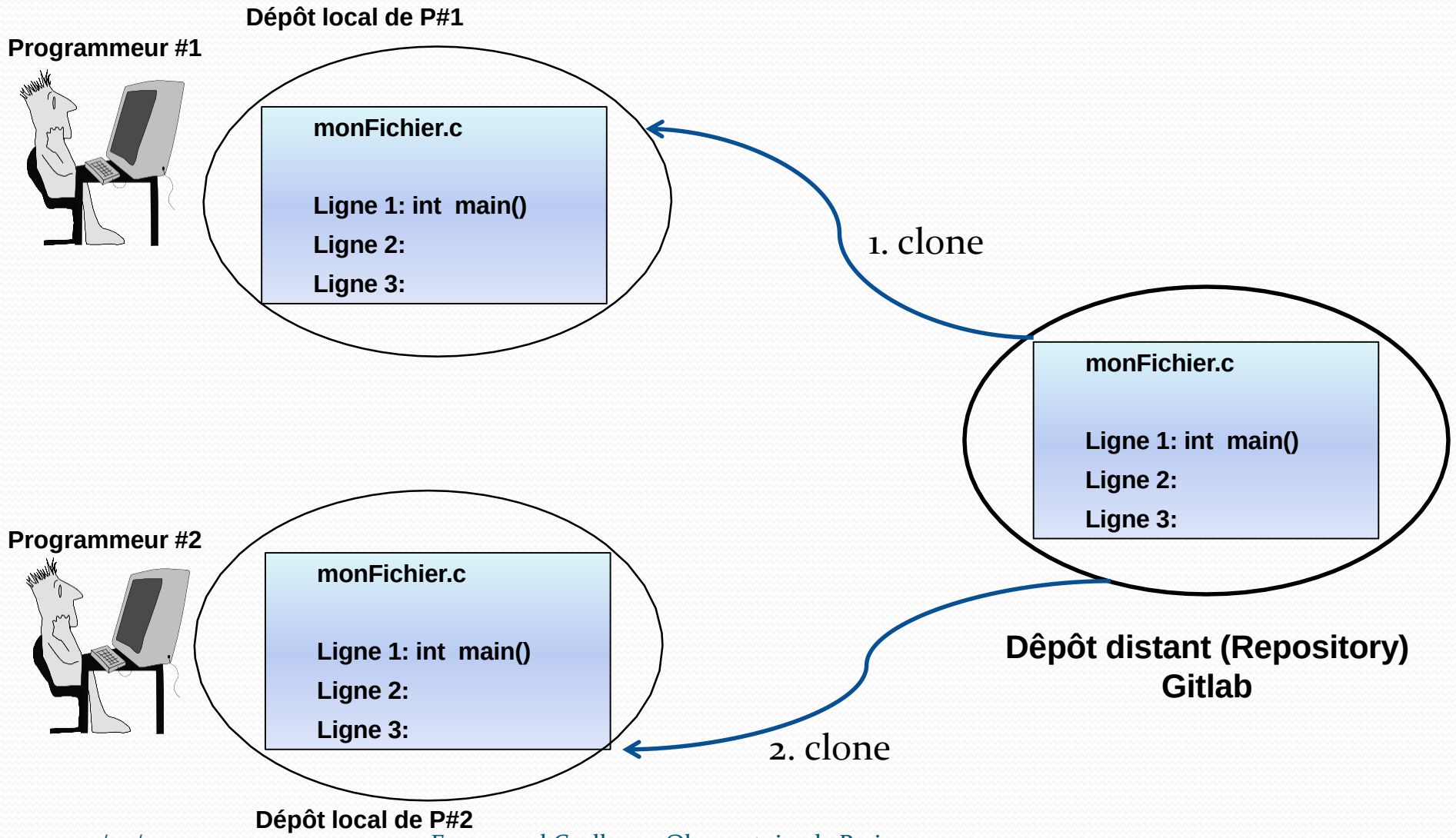
Les conflits



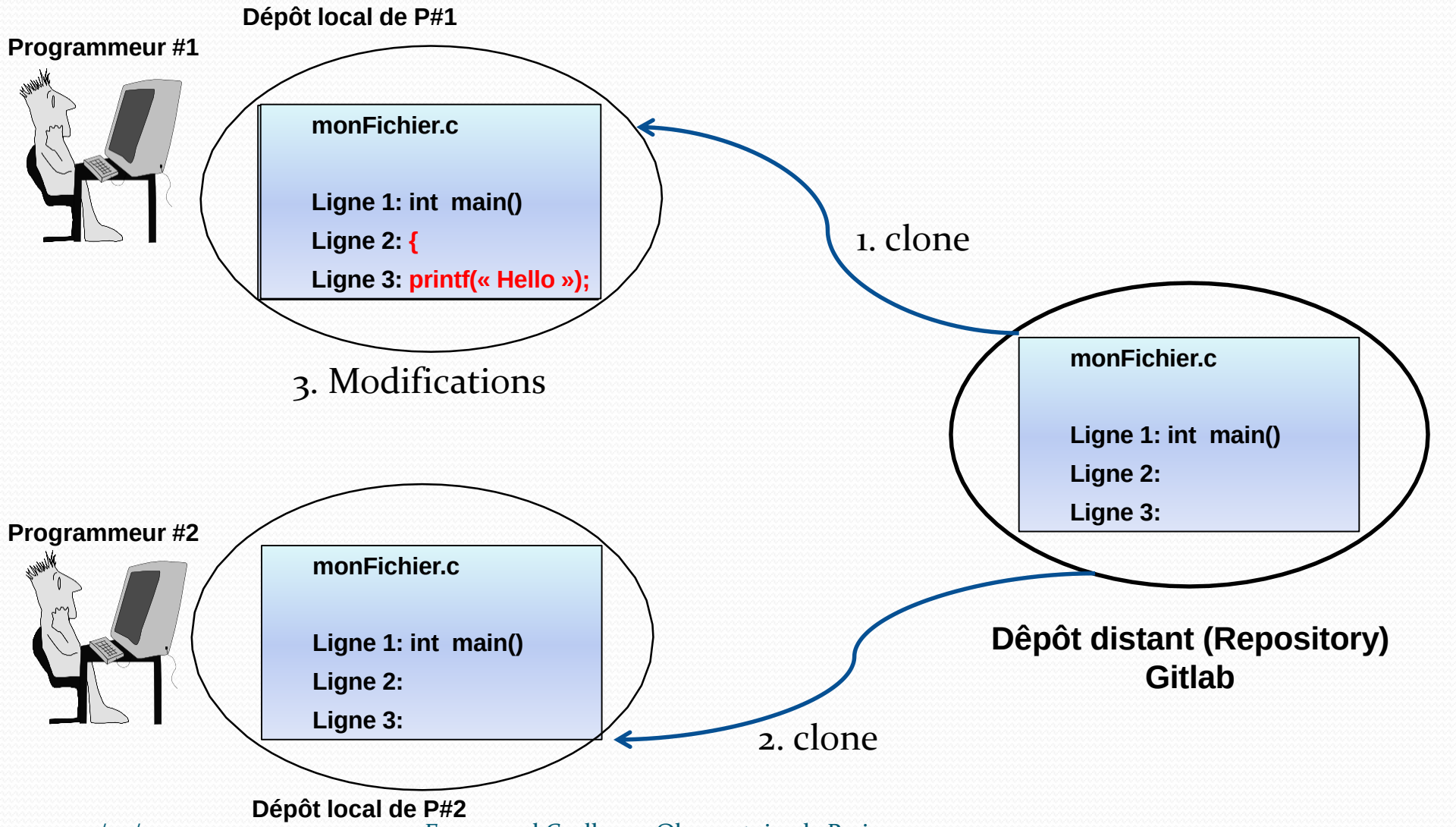
Les conflits



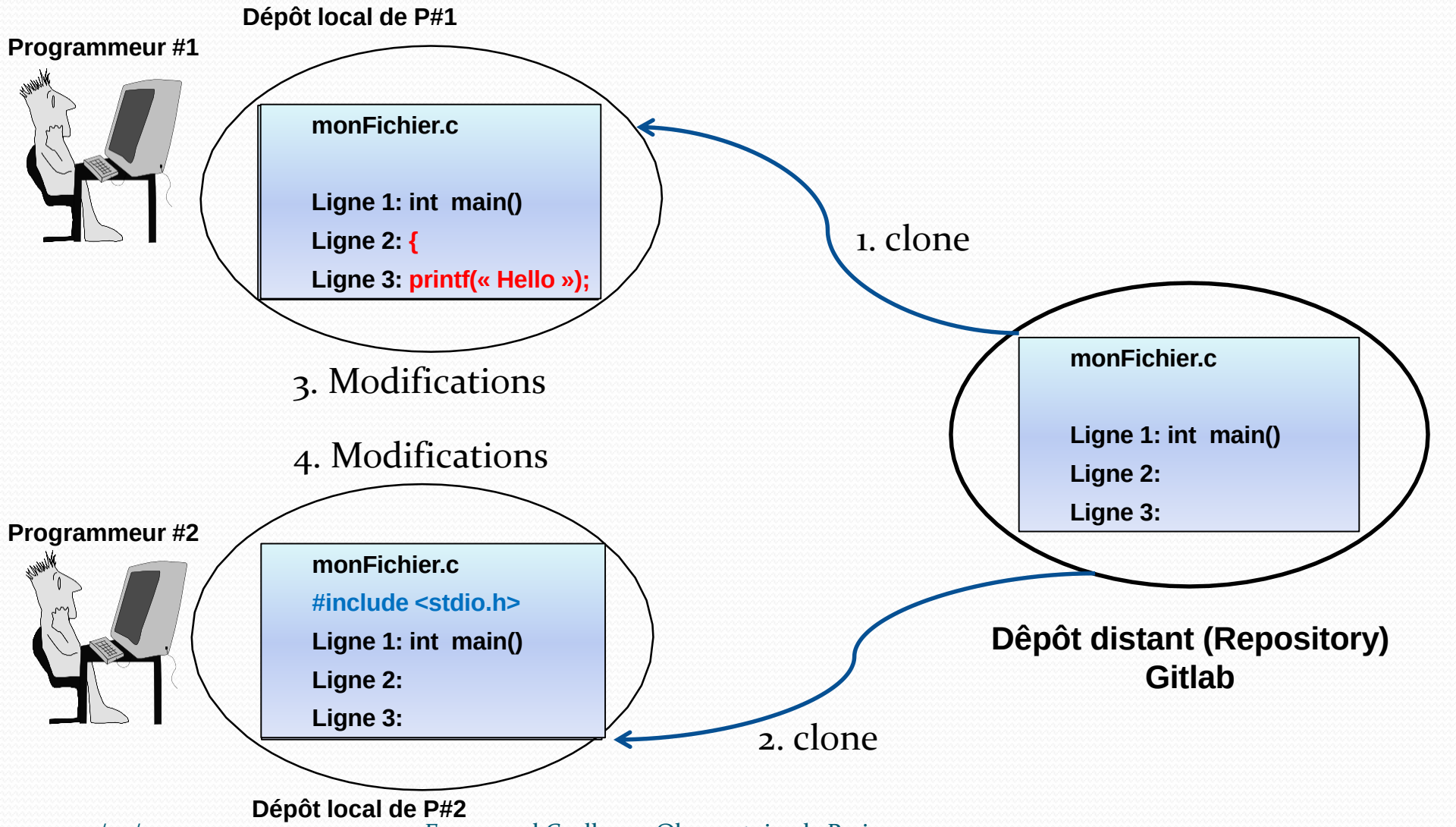
Les conflits



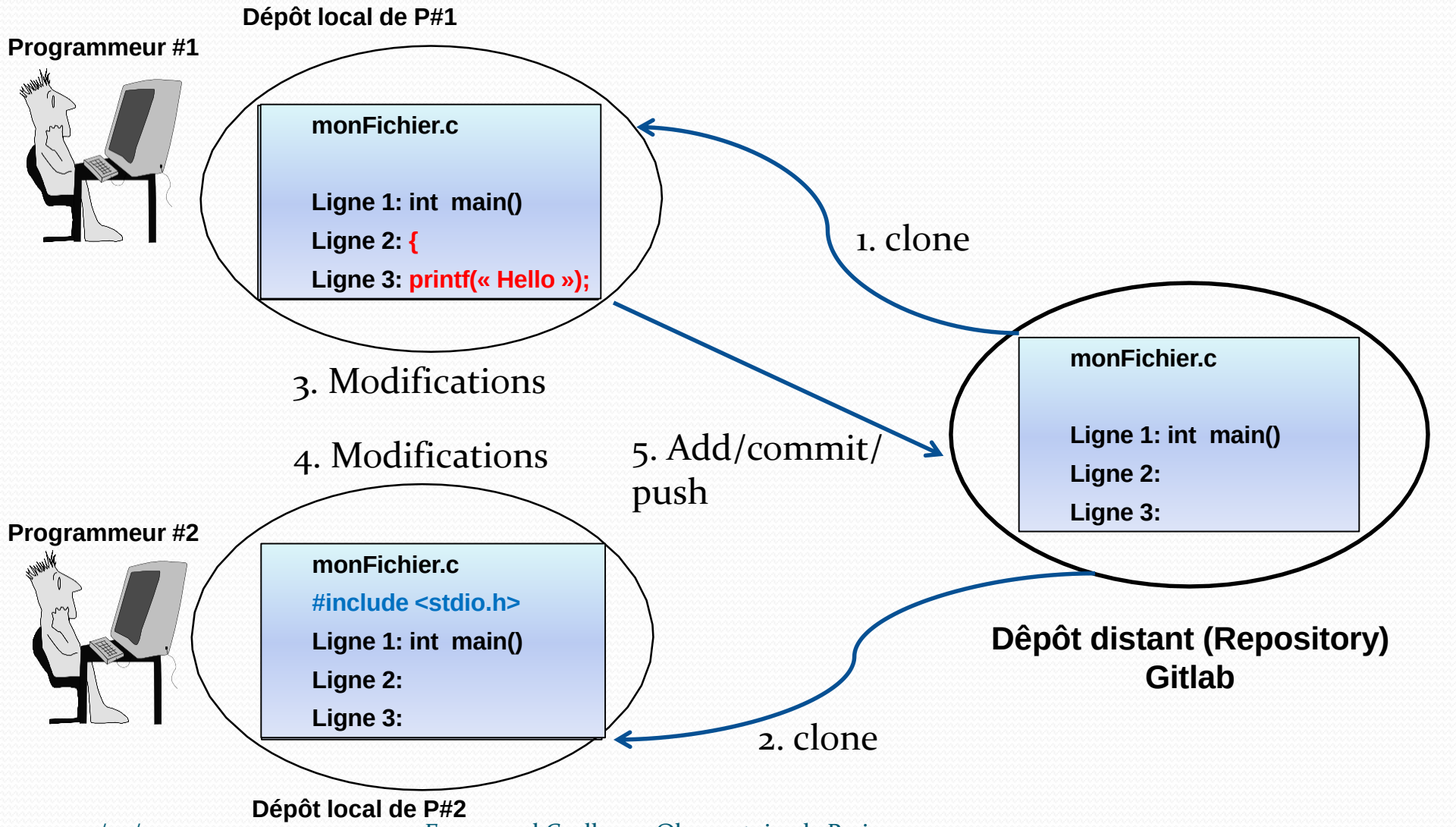
Les conflits



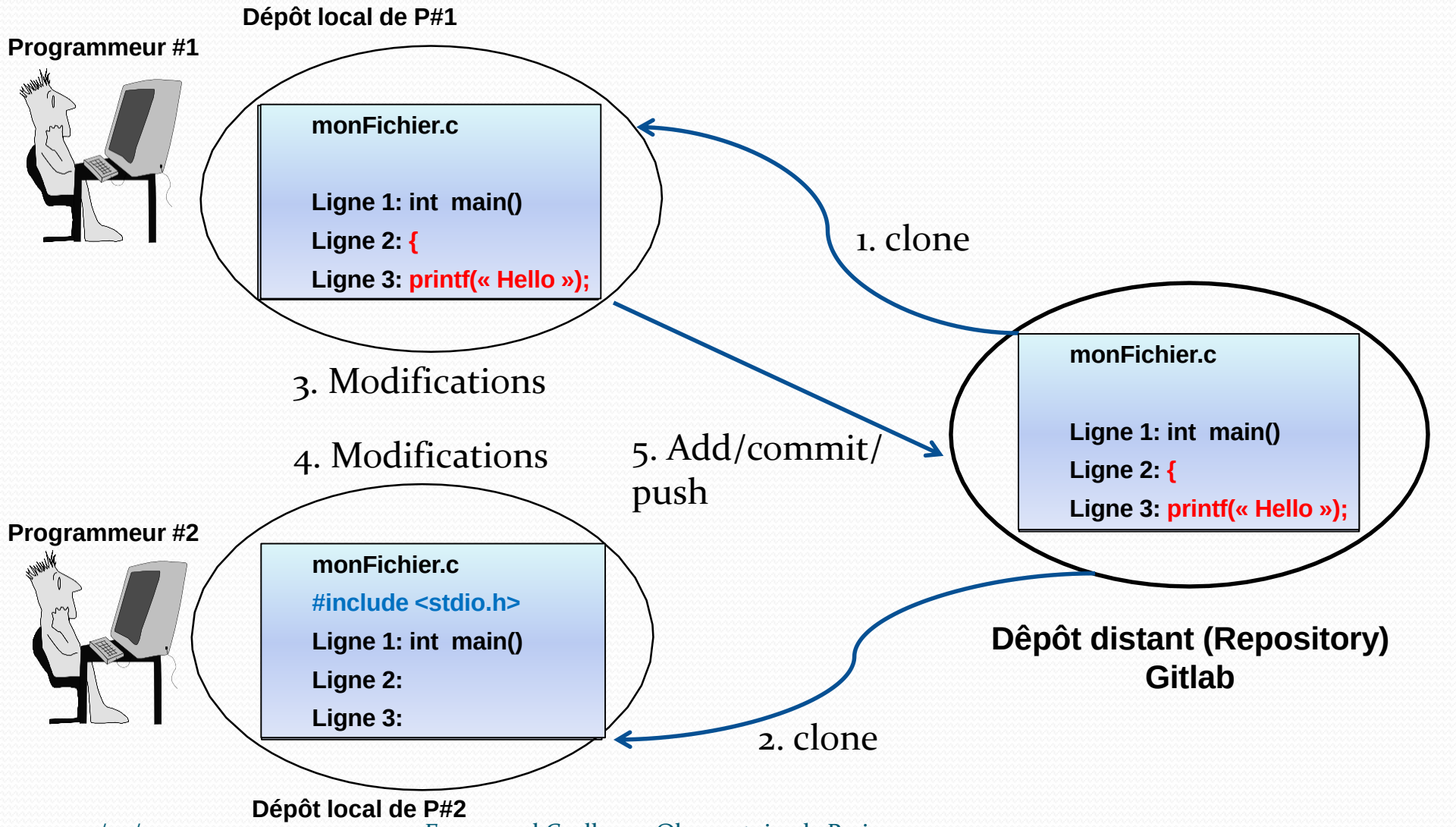
Les conflits



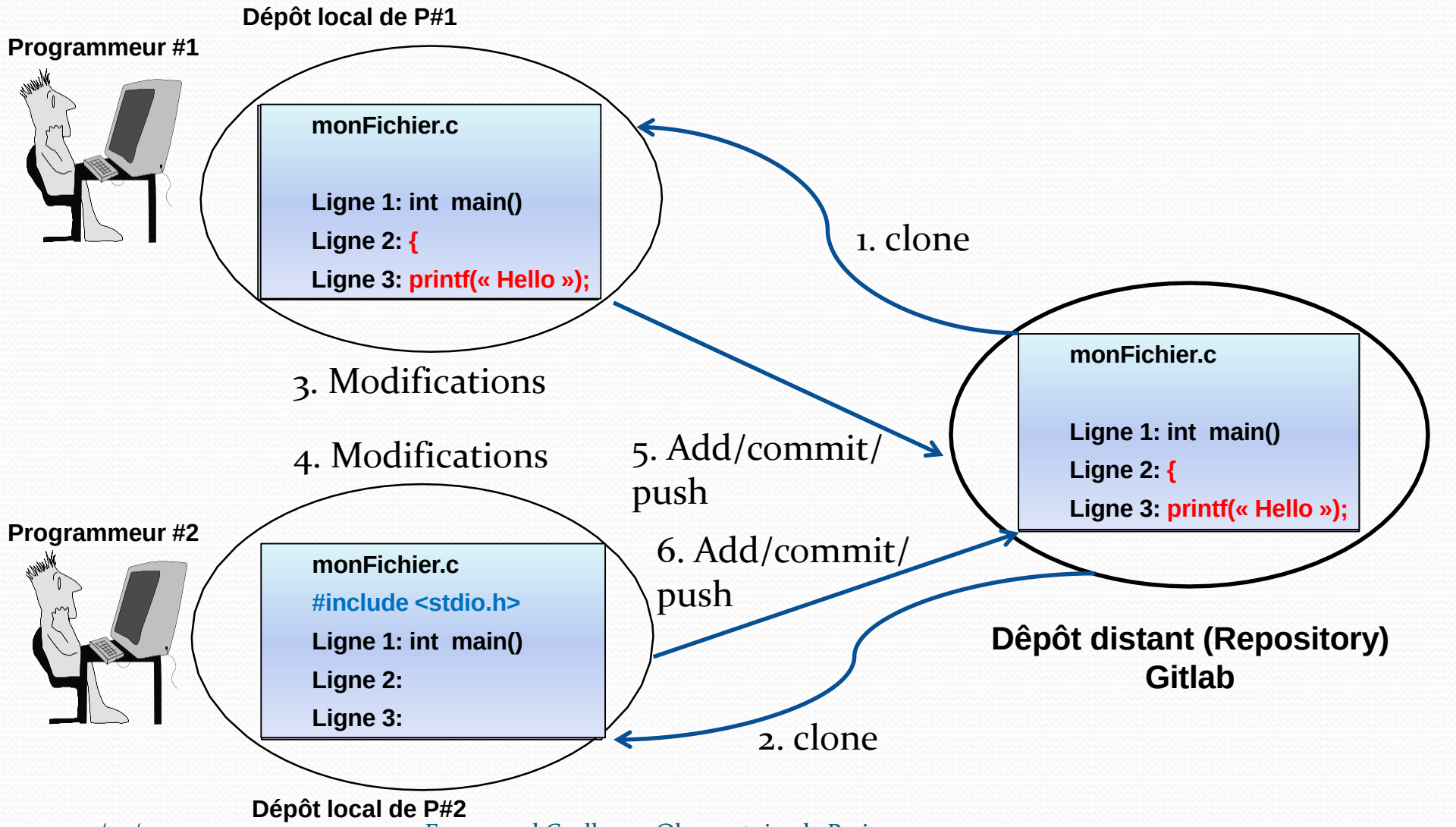
Les conflits



Les conflits



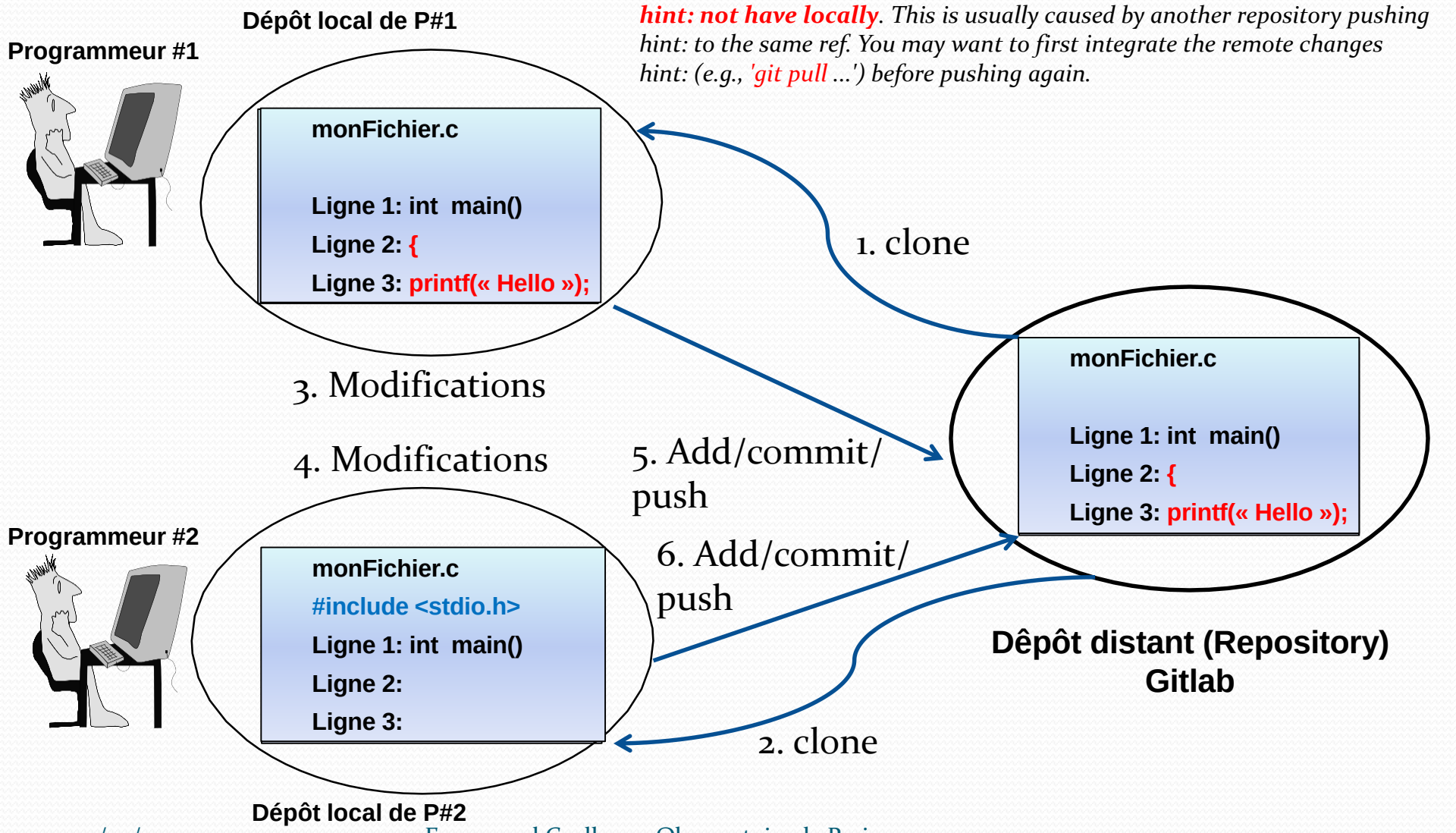
Les conflits



Les conflits

Git : CONFLIT !

error: failed to push some refs to 'https://CCC@ibgit.com:8443/scm/xxx.git'
*hint: Updates were rejected because **the remote contains work that you do***
***hint: not have locally**. This is usually caused by another repository pushing*
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.



Les conflits

Dépôt local de P#1

Programmeur #1

monFichier.c

Ligne 1: int main()
Ligne 2: {
Ligne 3: printf(« Hello »);

Programmeur #2

monFichier.c
#include <stdio.h>
Ligne 1: int main()
Ligne 2:
Ligne 3:

Dépôt local de P#2

monFichier.c

Ligne 1: int main()
Ligne 2: {
Ligne 3: printf(« Hello »);

Dêpôt distant (Repository) Gitlab

Les conflits

Dépôt local de P#1

Programmeur #1

monFichier.c

Ligne 1: int main()
Ligne 2: {
Ligne 3: printf(« Hello »);

monFichier.c

Ligne 1: int main()
Ligne 2: {
Ligne 3: printf(« Hello »);

Dêpôt distant (Repository) Gitlab

7. Pull

Programmeur #2

monFichier.c

#include <stdio.h>
Ligne 1: int main()
Ligne 2:
Ligne 3:

Dépôt local de P#2

Les conflits

Dépôt local de P#1

Programmeur #1

monFichier.c

Ligne 1: int main()
Ligne 2: {
Ligne 3: printf(« Hello »);

8. Gestion des conflits

error: Your local changes to the following files would be overwritten by merge:
xxx.c
Please commit your changes or **stash** them before you merge.

7. Pull

monFichier.c

Ligne 1: int main()
Ligne 2: {
Ligne 3: printf(« Hello »);

Dêpôt distant (Repository) Gitlab

Programmeur #2

monFichier.c

#include <stdio.h>
Ligne 1: int main()
Ligne 2:
Ligne 3:

Dépôt local de P#2

Les conflits

Dépôt local de P#1

Programmeur #1

monFichier.c

Ligne 1: int main()
Ligne 2: {
Ligne 3: printf(« Hello »);

8. Gestion des conflits

error: Your local changes to the following files would be overwritten by merge:
xxx.c
Please commit your changes or **stash** them before you merge.

7. Pull

monFichier.c

Ligne 1: int main()
Ligne 2: {
Ligne 3: printf(« Hello »);

Dêpôt distant (Repository)
Gitlab

Programmeur #2

monFichier.c

Ligne 1: int main()
Ligne 2:
Ligne 3:

9. Stash

Dépôt local de P#2

Les conflits

Dépôt local de P#1

Programmeur #1

monFichier.c

Ligne 1: int main()
Ligne 2: {
Ligne 3: printf(« Hello »);

8. Gestion des conflits

error: Your local changes to the following files would be overwritten by merge:
xxx.c
Please commit your changes or **stash** them before you merge.

7. Pull

10. pull

monFichier.c

Ligne 1: int main()
Ligne 2: {
Ligne 3: printf(« Hello »);

Dêpôt distant (Repository)
Gitlab

Programmeur #2

9. Stash

monFichier.c

Ligne 1: int main()
Ligne 2: {
Ligne3:printf(« Hello»);

Dépôt local de P#2

Les conflits

Dépôt local de P#1

Programmeur #1

monFichier.c

Ligne 1: int main()
Ligne 2: {
Ligne 3: printf(« Hello »);

8. Gestion des conflits

error: Your local changes to the following files would be overwritten by merge:
xxx.c
Please commit your changes or **stash** them before you merge.

7. Pull

10. pull

monFichier.c

Ligne 1: int main()
Ligne 2: {
Ligne 3: printf(« Hello »);

Dêpôt distant (Repository)
Gitlab

Programmeur #2

monFichier.c

#include <stdio.h>
Ligne 1: int main()
Ligne 2: {
Ligne3:printf(« Hello»);

9. Stash
11. Stash pop

Dépôt local de P#2

Les conflits

Dépôt local de P#1

Programmeur #1

monFichier.c

Ligne 1: int main()
Ligne 2: {
Ligne 3: printf(« Hello »);

8. Gestion des conflits

error: Your local changes to the following files would be overwritten by merge:
xxx.c
Please commit your changes or **stash** them before you merge.

7. Pull

monFichier.c
#include <stdio.h>
Ligne 1: int main()
Ligne 2: {
Ligne3:printf(« Hello»);

10. pull

Dêpôt distant (Repository)
Gitlab

12. Add/Commit/push

Programmeur #2

monFichier.c
#include <stdio.h>
Ligne 1: int main()
Ligne 2: {
Ligne3:printf(« Hello»);

9. Stash
11. Stash pop

Dépôt local de P#2

Les conflits

Dépôt local de P#1

Programmeur #1

```
monFichier.c
#include <stdio.h>
Ligne 1: int  main()
Ligne 2:  {
Ligne3:printf(« Hello»);
```

13. Pull

8. Gestion des conflits

error: Your local changes to the following files would be overwritten by merge:
xxx.c
Please commit your changes or **stash** them before you merge.

```
monFichier.c
#include <stdio.h>
Ligne 1: int  main()
Ligne 2:  {
Ligne3:printf(« Hello»);
```

7. Pull

10. pull

Dêpôt distant (Repository)
Gitlab

12. Add/Commit/push

Programmeur #2

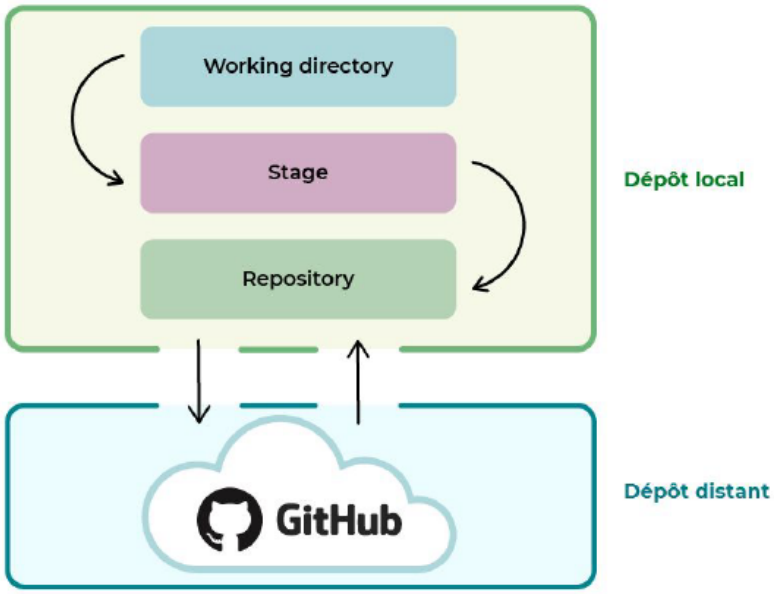
```
monFichier.c
#include <stdio.h>
Ligne 1: int  main()
Ligne 2:  {
Ligne3:printf(« Hello»);
```

9. Stash
11. Stash pop

Dépôt local de P#2

Fiche récap : Gérez du code avec Git et Github

Développement



Gestion des fichiers et des commits

git status
Pour montrer l'état des fichiers

$ git add fichier.html
Pour ajouter des fichiers à l'index pour le prochain commit

git commit -m "Message de commit"
Pour créer un nouveau commit avec les fichiers ajoutés à l'index

Gestion des branches

git branch
Pour lister toutes les branches

git branch NOM_DE_LA_BRANCHE
Pour créer une nouvelle branche

git checkout NOM_DE_LA_BRANCHE
Pour changer de branche

git merge NOM_DE_LA_BRANCHE
Pour fusionner la branche spécifiée dans la branche actuelle

Configuration et initialisation

$ cd Documents/PremierProjet
Pour vous positionner dans le dossier PremierProjet

$ git init
Pour initialiser un nouveau dépôt Git

git clone URL_DU_REPO
Pour cloner un dépôt existant à partir de l'URL fournie

Travailler avec des repos distant

git push
Pour envoyer la nouvelle version sur le dépôt distant

git pull
Pour récupérer les dernières modifications du dépôt distant

Historique et inspection

git log
Pour voir l'historique des commits

git stash
Pour enregistrer temporairement des modifications non indexées

git stash apply
Pour appliquer les modifications enregistrées avec stash

git status

On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

modified: simu_v13/README.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

modified: simu_v14/README.md

git tag

- La commande git tag permet de « taguer » l'ensemble des fichiers dans une version donnée pour leur attribuer un label commun, par exemple version 1.2

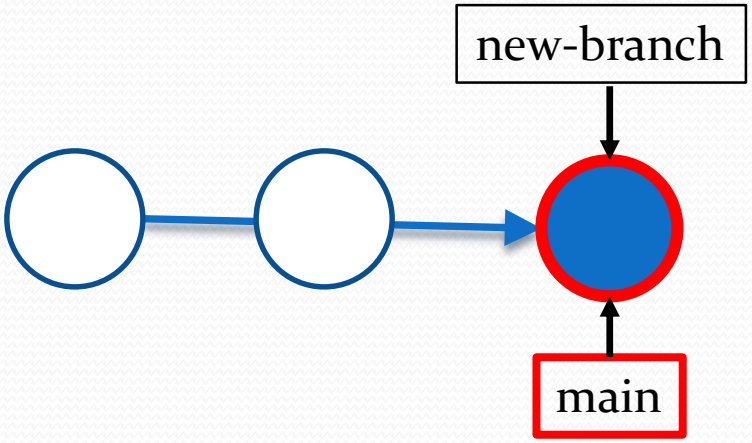
- git tag
  - liste des tags
- git tag 1.2
  - créer un tag simple « 1.2 » pour l'ensemble des fichiers du projet
- git tag 1.2 -m « Version 1.2 »
  - créer un tag annoté « 1.2 » pour l'ensemble des fichiers du projet
- git show 1.2
  - Information sur le contenu du tag

git branch

- La commande **git branch** permet de lister, créer, supprimer une branche de code.
- git branch
  - liste des branches.
    - * main

    l'étoile signifie que c'est la branche que nous utilisons actuellement.
- git branch new-branch
  - créer une nouvelle branche de nom new-branch en local à partir de la branche courante.
  - Attention, on travaille toujours sur la branche courante, nous n'avons pas encore basculé sur la nouvelle branche

git branch

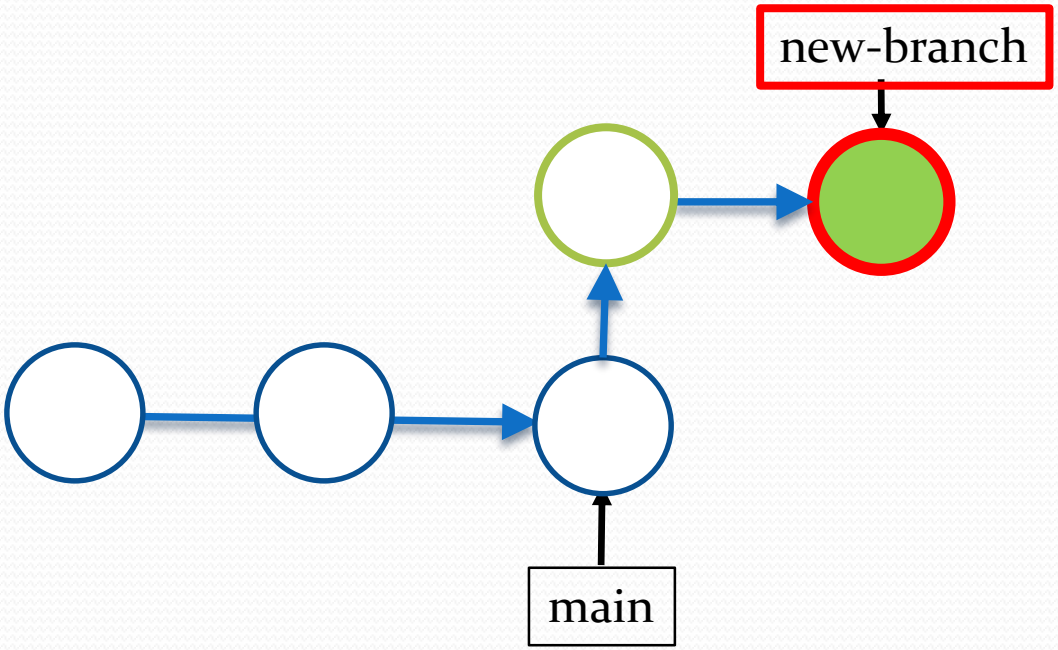



```
$git branch
   * main
     new-branch
```

git checkout

- git checkout new-branch
  - Se déplacer sur notre branche locale new-branch
- Alternative: git checkout -b new-branch
  - Créer une nouvelle branche new-branch et se déplacer directement dessus
- git fetch + git checkout branch_distante
  - Récupérer une branche distante et se déplacer dessus

git branch / checkout

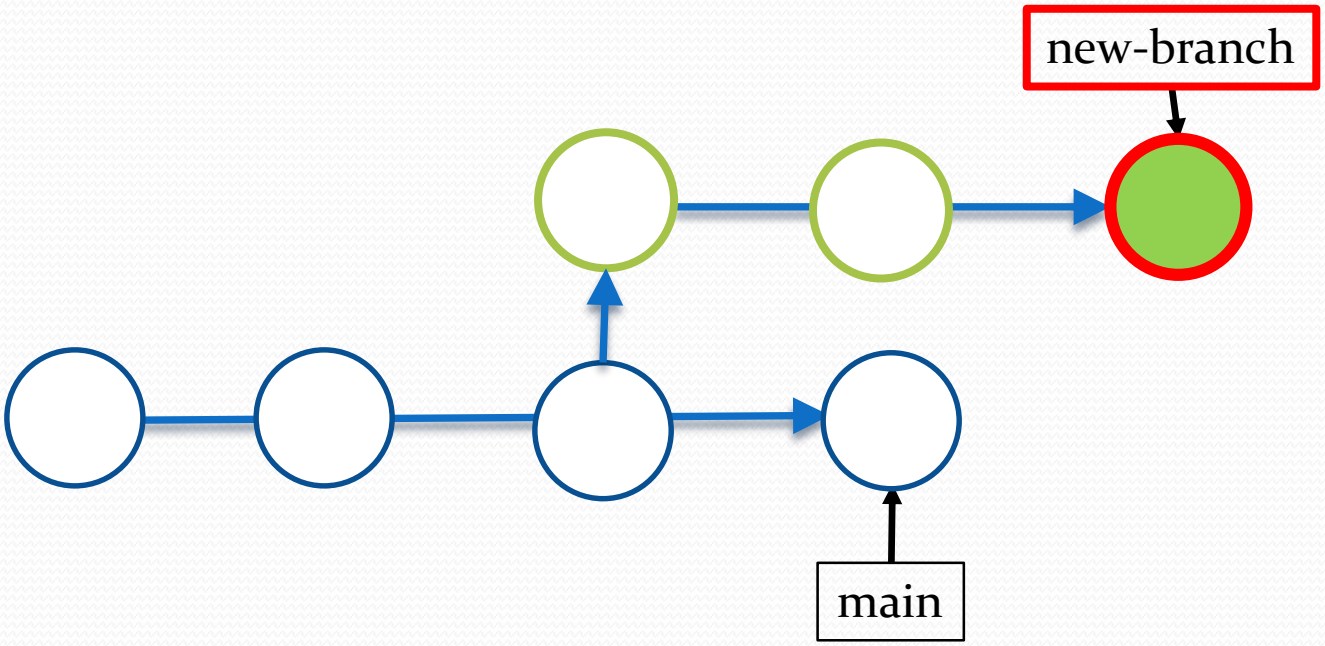


$git checkout –b new-branch
$git branch
main
* new-branch

git merge

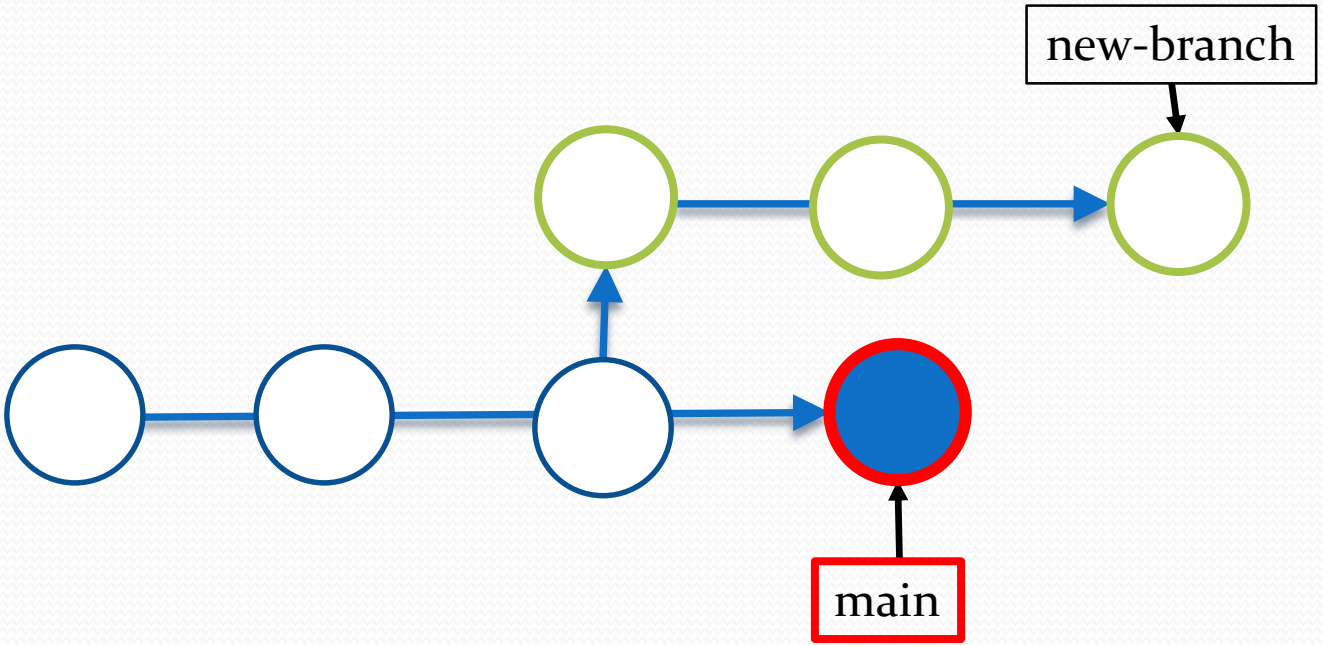
- git merge new-branch
  - Fusionner, dans la branche courante, la branche new-branch
  - Il faut donc d'abord se déplacer dans la branche main pour fusionner nos modifications réalisées dans new-branch

git merge



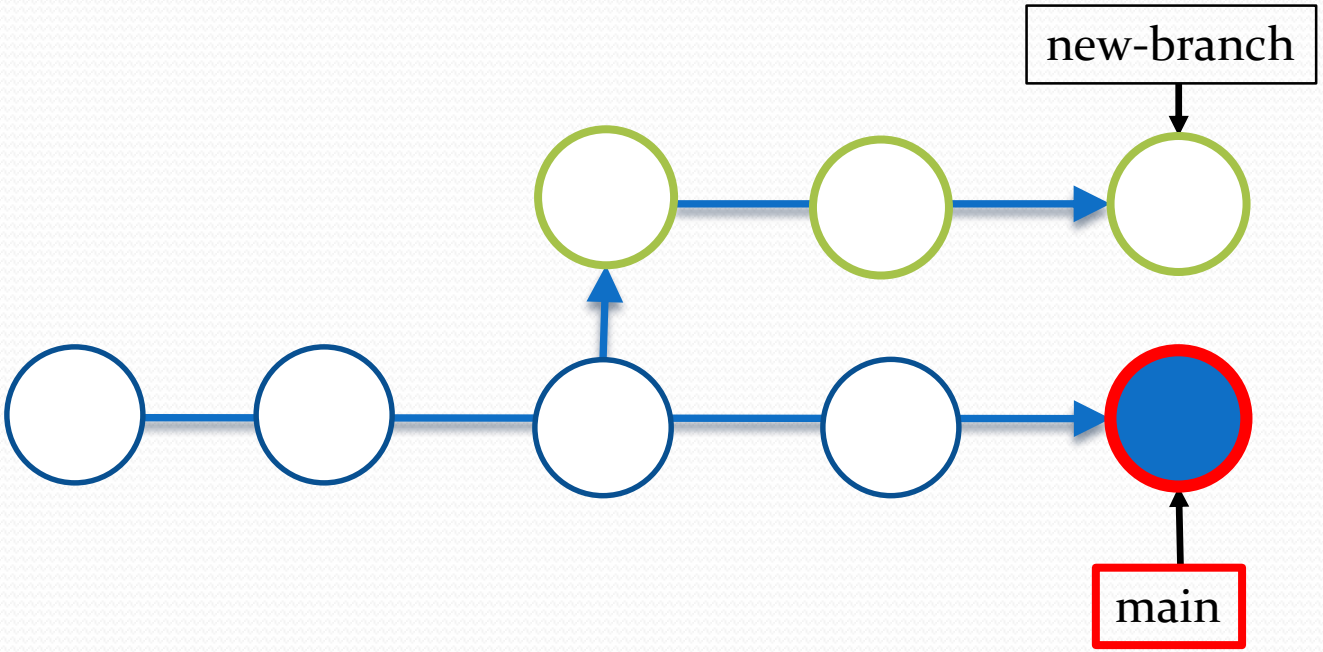

```
$git branch
      main
  *  new-branch
```

git merge



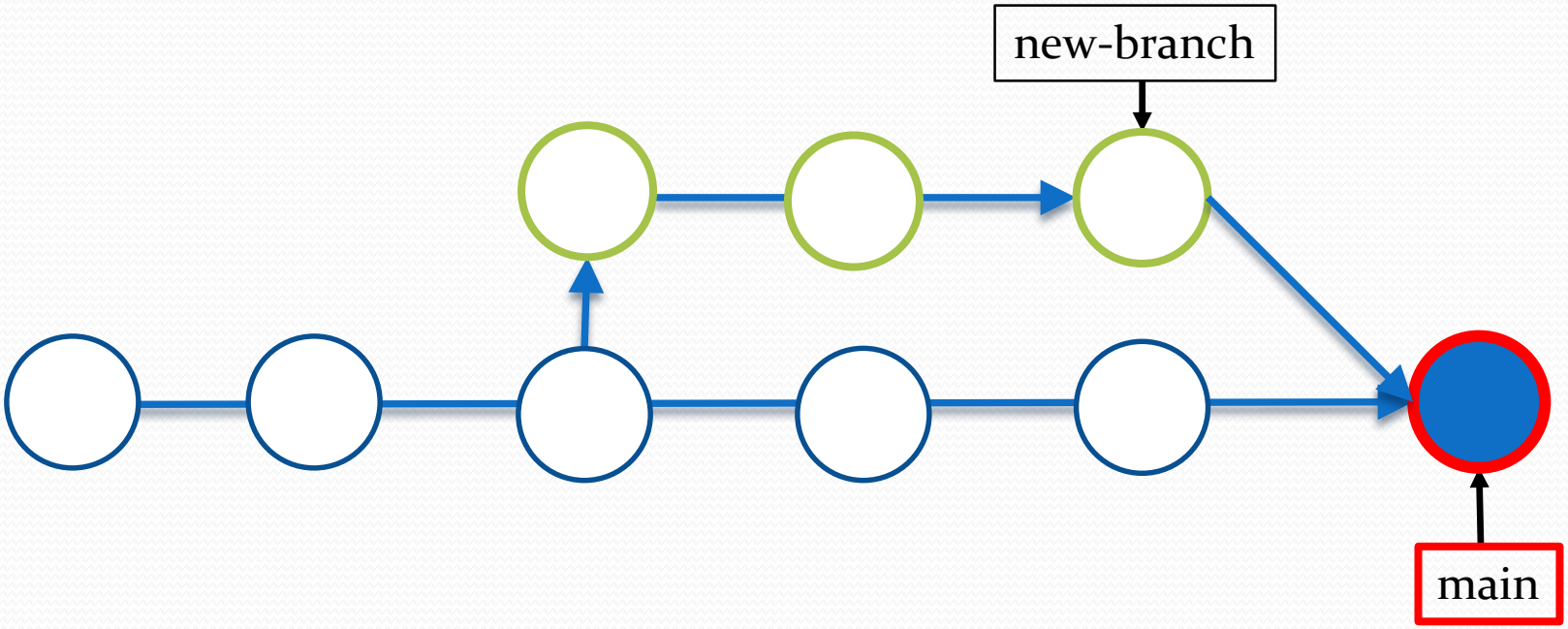
$git checkout main
$git branch
  * main
    new-branch

git merge



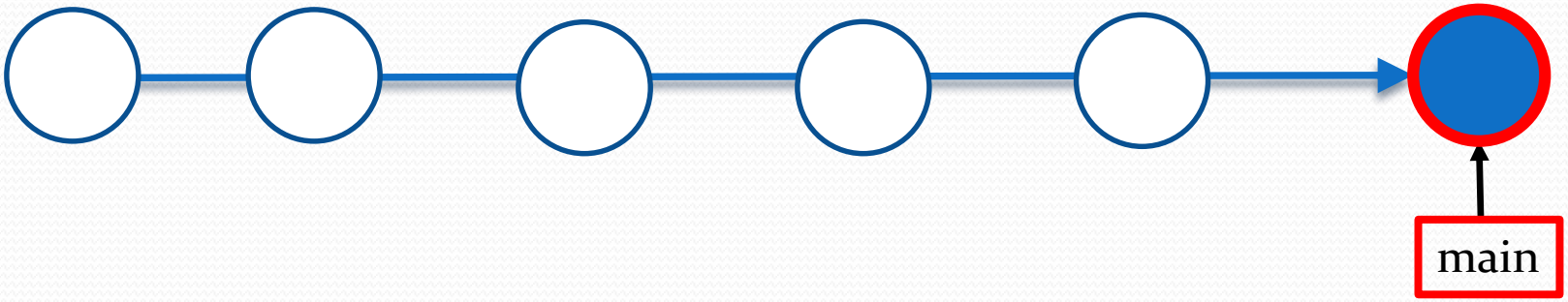
$git pull

git merge



$git merge new-branch

git merge



$git branch -d new-branch

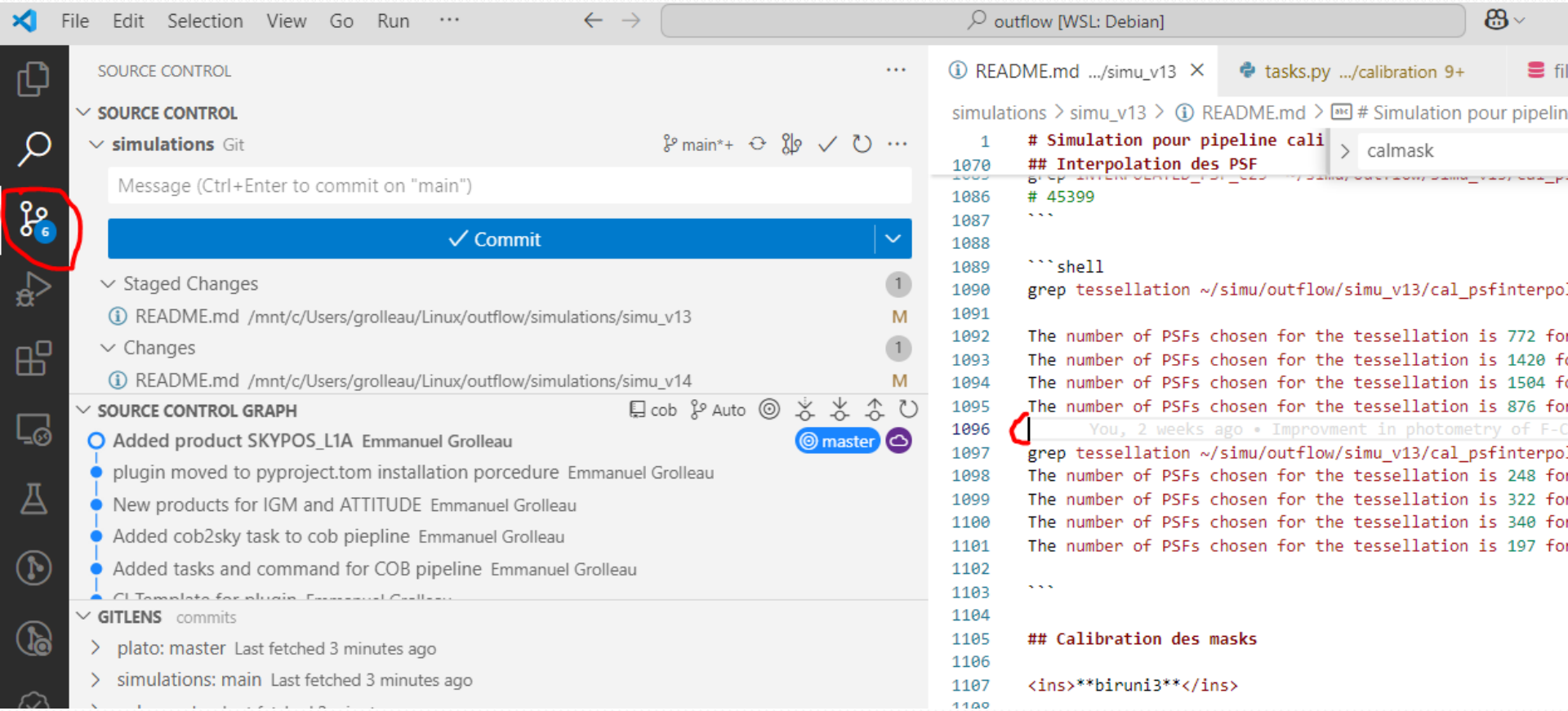
Supprimer également la branche sur le serveur distant
$git push origin -d new-branch

Bonnes pratiques Git

- Interactions avec le dépôt
  - Un « pull » doit toujours précéder un « push ».
  - Mettre à jour l'ensemble de la copie locale.
  - Mettre à jour la copie locale périodiquement.
  - Ne versionner que les fichiers nécessaires au projet.
  - Un « commit » doit représenter un tout.
  - Chaque « commit » doit comporter un message représentatif de l'objectif et de la raison des modifications envoyées et non un résumé des modifications

Intégration de Git dans les IDE

Intégration de Git dans Visual Studio Code



Sources

- Git Manuel :
  - https://git-scm.com/docs/user-manual
- Liste des clients :
  - https://git-scm.com/downloads/guis
- Cours :
  - https://openclassrooms.com/courses/gerer-son-code-avec-git-et-github