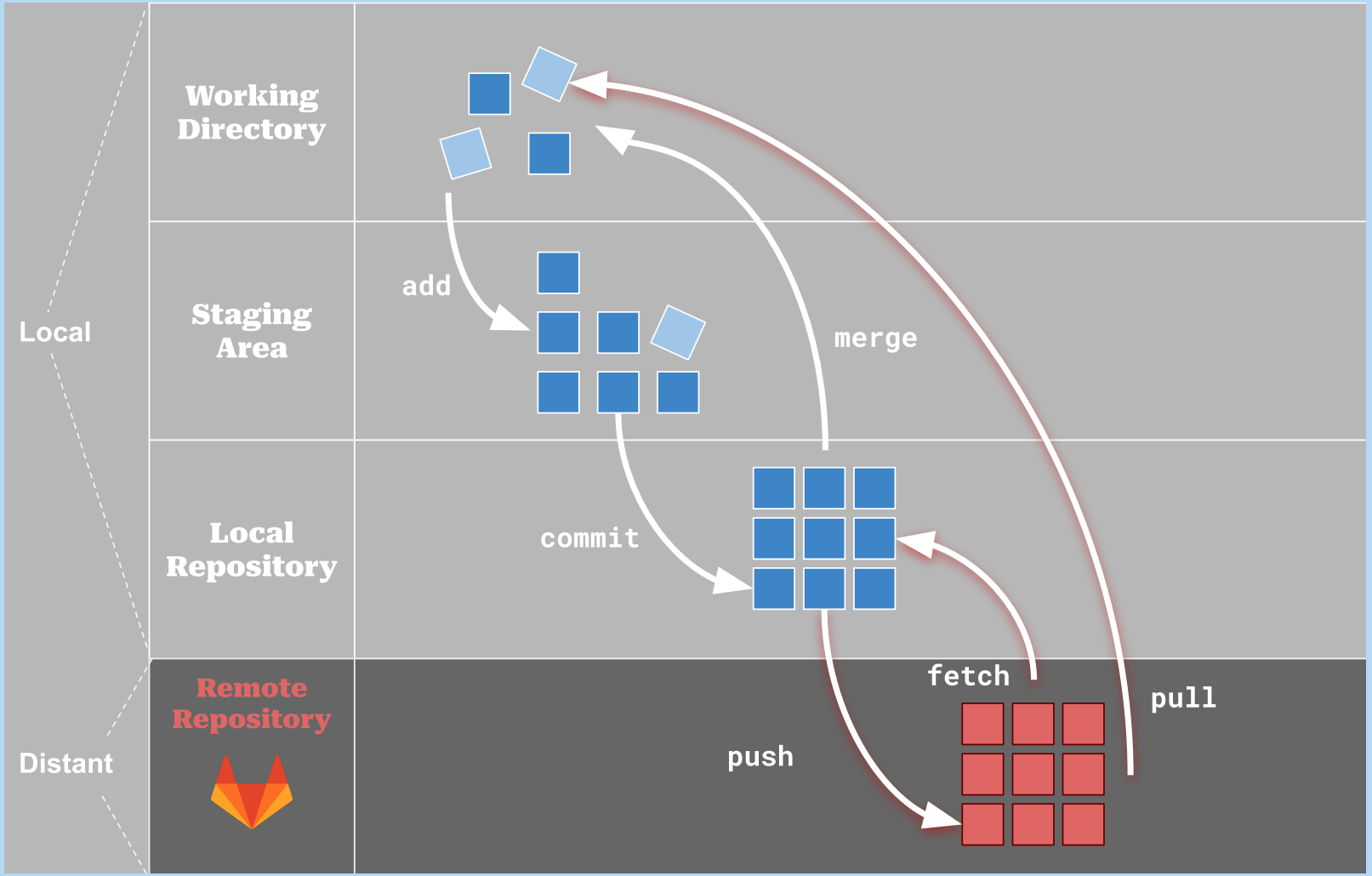
Git est défini comme un outil permettant le contrôle des versions d'un projet.
DISTRIBUTED VERSION CONTROL SYSTEM

- **Version Control:** Chaque étape de développement du projet est mémorisé, comme pris en photo.
- **Distribué:** Chaque dépôt, sur chaque machine est autonome. Pas besoin d'être connecté pour travailler.
- **Collaboratif:** Lorsque plusieurs personnes souhaitent collaborer sur un même projet, l'utilisation d'un dépôt de référence permet de gérer les contributions de chacun : Github / Gitlab par exemple
- **Flexible:** De tout petits à de très larges projets, mono ou multi contributeurs

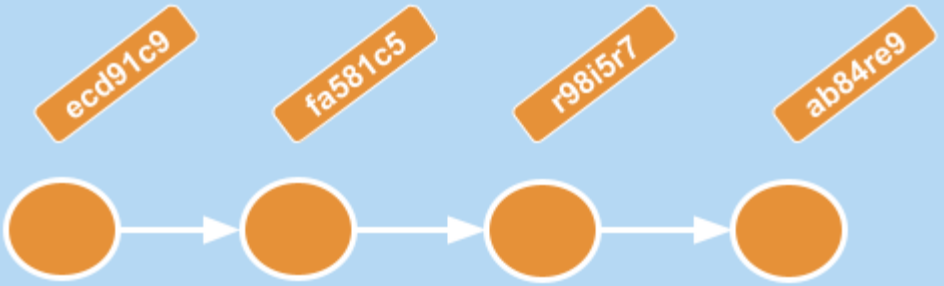
VOCABULAIRE

- **Répertoire de travail :** Les fichiers que l’on manipule et qui correspondent à une extraction unique d’une version du projet
- **Index, staging area :** désigne tous les fichiers modifiés que vous souhaitez voir apparaître dans votre prochain commit.
- **Repository local :** référentiel qui stocke les commits et les autres informations qui sont utiles à la gestion de versions du projet
- **Repository distant :** référentiel distant (remote) qui sert à partager l'historique de commits ou à le synchroniser avec d’éventuels membres d’une équipe
- **origin :** nom abrégé du référentiel distant à partir duquel un projet a été cloné à l'origine


Local
Distant
Working Directory
Staging Area
Local Repository
Remote Repository
add
commit
merge
push
fetch
pull

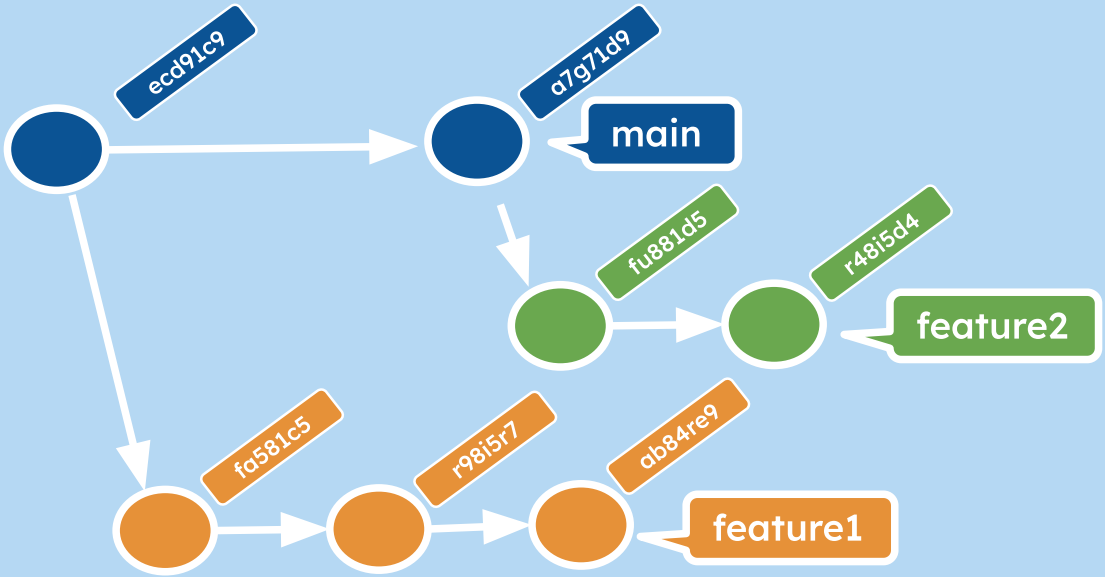
VOCABULAIRE - UN COMMIT

- référence qui permet de retrouver l'état du projet à un moment donné dans le temps
- inclut chaque fichier et dossier, avec l'ensemble des modifications qui ont été enregistrées dans l'historique du projet.
- identifiable par un hash (SHA-1). Exemple : ecd91c9baab1e7cab9dce65fa9dbefdaf13cd34d généralement résumé aux 7 premiers caractères : ecd91c9.



VOCABULAIRE - UNE BRANCHE

- Le chemin qui relie plusieurs commits
- La branche par défaut se nomme master (main) et est destinée, par convention, à être la branche principale du projet
- Permet le développement de différentes versions en parallèle
- 1 pointeur vers un commit (le plus récent)



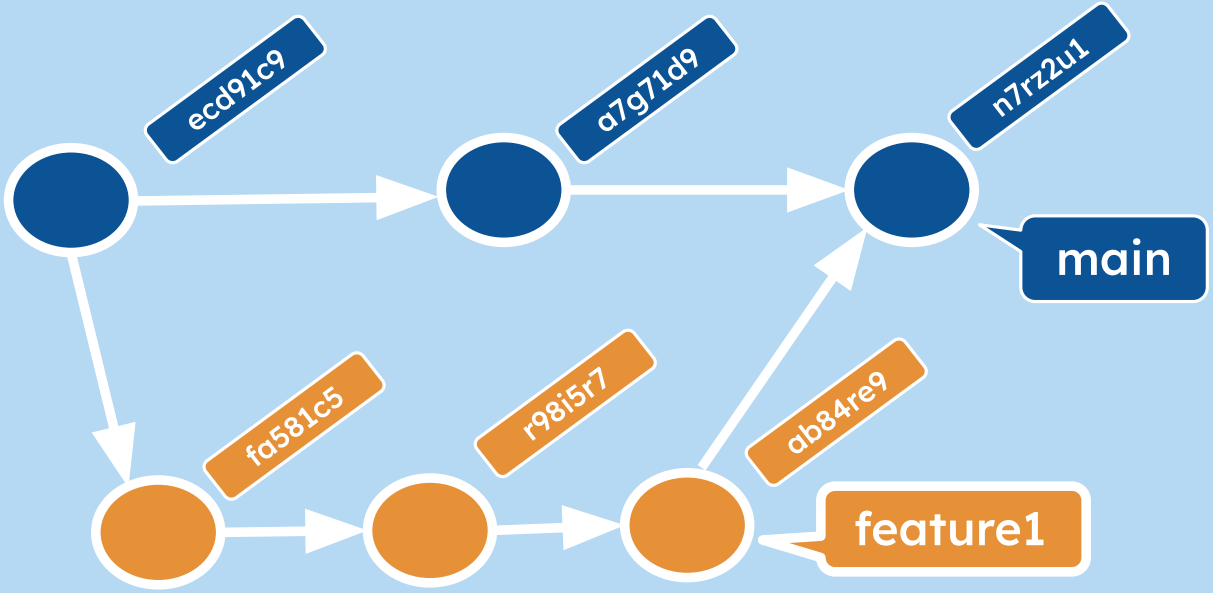
VOCABULAIRE - UN TAG

- Une étiquette/un alias identifiant un commit
- Permet la persistance d'un commit
- Principalement utilisé pour identifier les versions officielles d'un logiciel
- Evènement particulier du cycle utilisable comme déclencheur dans la CI/CD

```
$ git tag v1.3 -m "Première version livrée au client"
```

VOCABULAIRE - UN MERGE

- Fusionner 2 branches / Réconcilier 2 historiques
- Rapatrier les modifications d'une branche dans une autre



PREMIERS PAS AVEC GIT

- Git en Ligne de commande

```
$ git <command> <arguments>
$ git help
$ git help <command>
```

- Initialisation de votre identité

```
$git config --global user.name  "Votre nom"
$git config --global user.email  "votre @mail"
```

DÉMARRER UN PROJET

- Initialisation d’un dépôt Git dans un répertoire existant

Si le dossier de travail indiqué n’existe pas, git va le créer.
Sans paramètre, la commande crée un dépôt git pour le dossier courant

```
$ git init mon_projet
```

- Clone d’un dépôt existant à partir d'une url (git@ ou https)

```
$git clone git@gitlab.in2p3.fr:cylo/qualite-logicielle-devlog.git
$git clone https://gitlab.in2p3.fr/cylo/qualite-logicielle-devlog.git
```

LE FICHIER .GITIGNORE

- Git c'est très pratique pour partager du code, mais on ne veut pas toujours mettre à disposition toutes ses sources.
- Par exemple, le fichier qui comprend les mots de passe des bases de données, des fichiers caches ou compilés, etc.
- Git fournit bien évidemment un outil pour ça: le fichier `.gitignore`
- Ce fichier se nomme forcément .gitignore (Il commence donc par un point !).
- Il se trouve à la racine du repository

LE FICHIER .GITIGNORE

```
*.txt
dossier/
dossier2/*.jpg
motdepasse.csv
/ressources/*
!/ressources/presentations
```

Ligne	Effet
`*.txt`	Cette ligne permettra d'ignorer tous les fichiers textes où qu'ils soient
`dossier/`	Cette ligne ignorera l'ensemble du contenu de `dossier` et par extension, le dossier lui-même (Git ne conserve pas les dossiers vides)
`dossier2/*.jpg`	Cette ligne ignorera les *.jpg dans le dossier2. Par contre, si dossier2 a des enfants (dossier2/sousdossier1) et des jpg à l'intérieur, il seront versionnés
`motdepasse.csv`	Cet ligne permet d'ignorer le fichier motdepasse.csv dans le dossier principal
`/ressources/*/` `!/ressources/presentations`	Ces ligne permettent d'ignorer le contenu du répertoire ressources sauf ce qui se trouve dans presentations

MON PREMIER COMMIT

- `git status` permet à tout moment de connaître l'état actuel du dépôt

```
$ git status
 >> Sur la branche cyril_branch_demo  rien à valider, la copie de travail est propre
 $ echo "Ajout d'une ligne dans mon fichier" > monFichier.txt
 $ git status
 >> Sur la branche cyril_branch_demo
 >> Fichiers non suivis :
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)
  monFichier.txt
```

- `git add` permet de référencer le fichier physique dans l'index

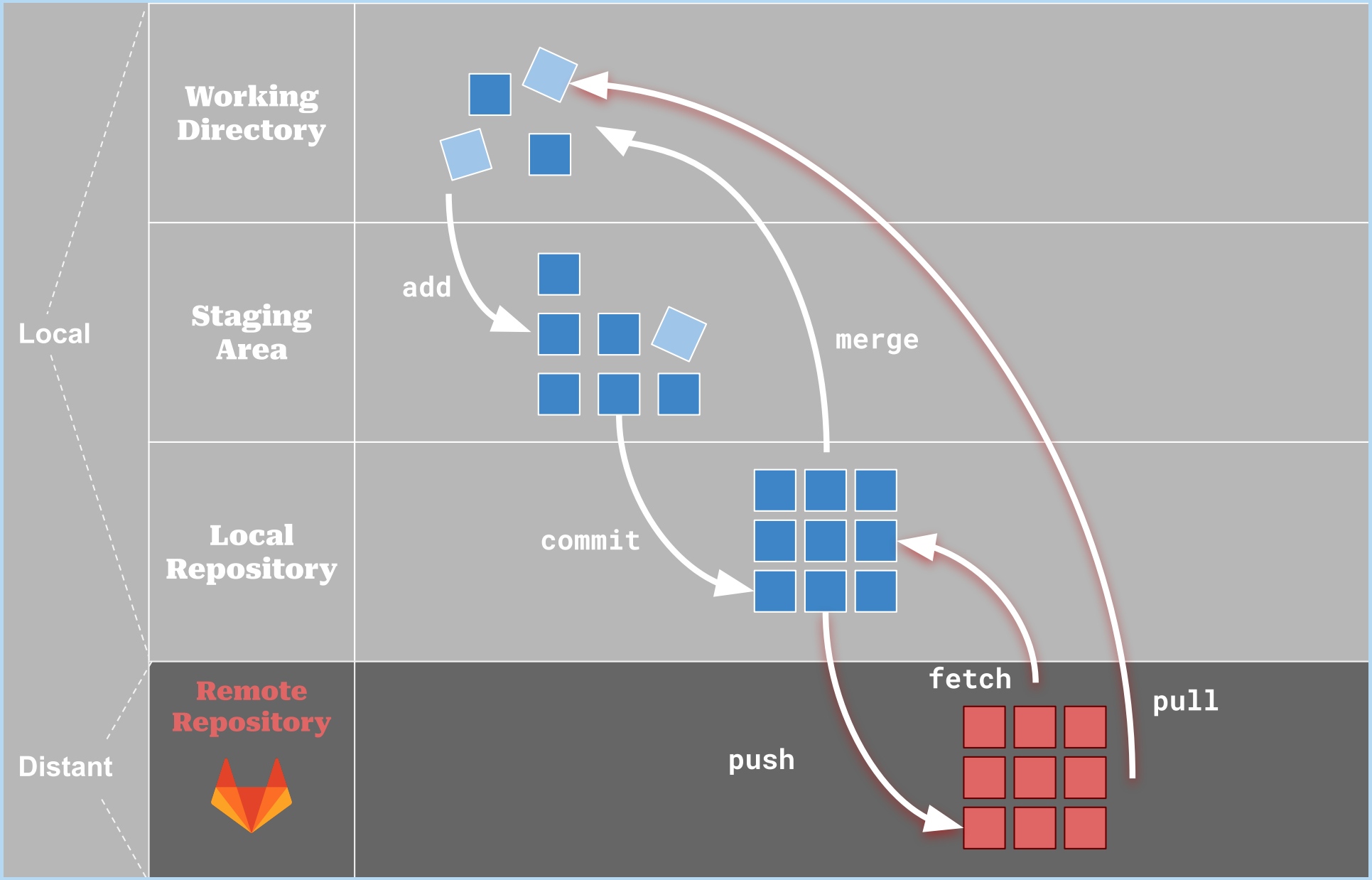
```
$ git add monFichier.txt
 $ git status
 >> Sur la branche cyril_branch_demo
 >> Modifications qui seront validées :
    (utilisez "git restore --staged <fichier>..." pour désindexer)
    nouveau fichier: monFichier.txt
```

- Valider le nouveau fichier et, ainsi, créer le commit : `git commit`

```
$ git commit -m “message pour expliquer ce que j’ai fait”
$ git status
>> Sur la branche cyril_branch_demo, rien à valider, la copie de travail est propre
```

BONNES PRATIQUES

- Faire des commits fréquents
- Ecrire des messages de commits clairs
- Un formalisme peut être adopté pour cette partie
  - [func] mon commentaire pour l'ajout d'une fonctionnalité
  - [edit] pour la modification d'une fonctionnalité
  - [del] pour la suppression d'une fonctionnalité ou fichier
  - [fix] pour la correction d'un bug
  - [refa] pour du refactor de code
  - [misc] quand aucun des tags précédant ne correspond à la tâche


Working Directory
Staging Area
Local Repository
Remote Repository
Local
Distant
add
commit
merge
push
fetch
pull

DÉPOT DISTANT

- Transférer les commits locaux vers le dépôt distant: `git push`

```
git push <remote> <branche>
git push <remote> --all # Permet d'envoyer toutes les branches
```

- Git ne permet pas de push si le dépôt distant est en décalage , l'option --force est nécessaire dans ce cas

```
git push <remote> --force
```

- Importer les informations du dépôt distant: `git fetch`

```
git fetch <remote> <branche>
git fetch <remote>  # Récupère toutes les branches et tous les commits
```

- `git pull` : cette commande récupère le contenu distant (fetch) puis tente de mettre à jour le contenu local par fusion (merge)
  - Dans une grosse collaboration il faut régulièrement utiliser cette commande pour éviter les conflits

```
git pull <remote> <branche>
git pull <remote>
```

DÉPOT DISTANT : GITLAB

- Pour cloner / pull / push un projet sur gitlab 2 protocoles possibles : HTTPS et SSH
- l'avantage principale de la clé SSH avec la sécurité est le fait de ne pas re-saisir son login/mdp à chaque opération distante

La configuration pour le ssh se déroule en 2 temps

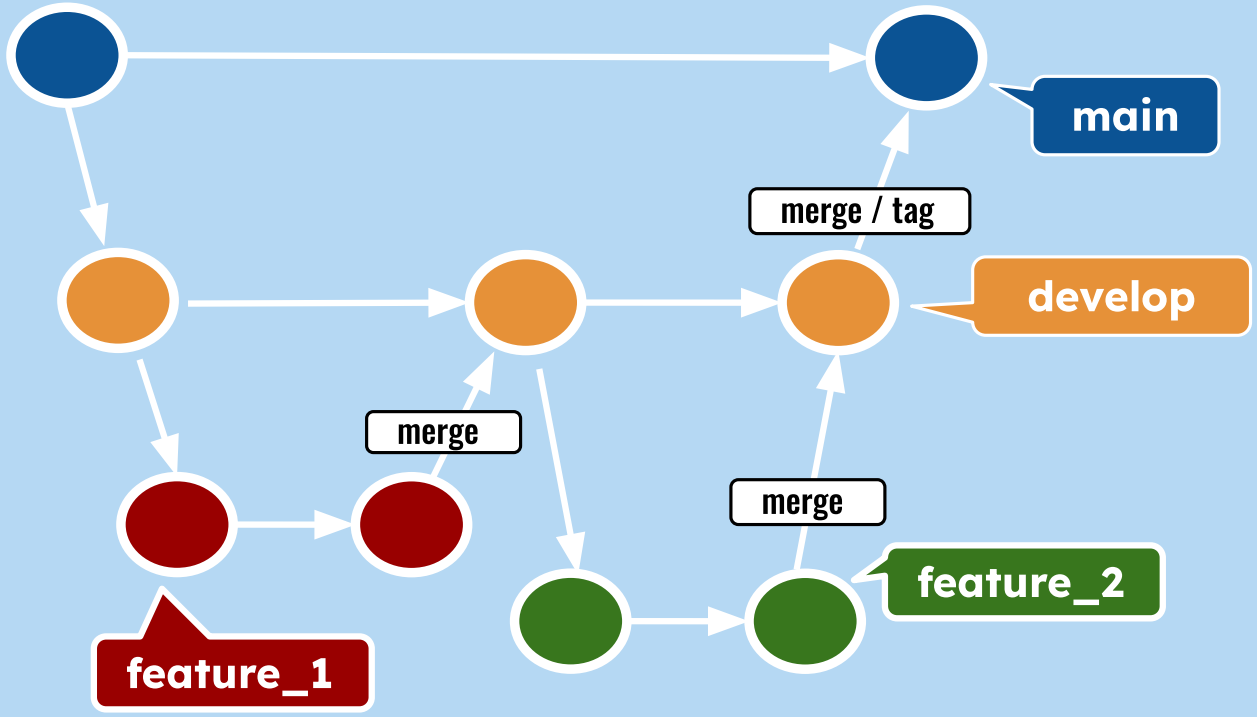
1. Génération ou récupération d'une clé SSH
2. Ajout de la clé dans Gitlab : https://gitlab.in2p3.fr/-/profile/keys

UNE COLLABORATION BASÉ SUR LES BRANCHES

- Interagir sur un dépot distant : la base pour un outil collaboratif
- Par contre interagir sur la même branche peut poser problème
  - on peut vouloir intégrer des fonctionnalités différentes et les tester individuellement
  - on peut vouloir faire des tests de développement
  - on peut vouloir garder une branche de référence propre et testée
- Il est donc nécessaire de créer des branches et de bien les organiser

BONNES PRATIQUES

- Au départ d'un projet il existe une seule branche : master (ou main)
- Idéalement on ne fait jamais de `git push` sur cette branche
- 1 branche de référence : exemple develop
- 1 branche = 1 unité de développement



LES BRANCHES

- Ajout, listing, suppression, renommage : `git branch`

```
git branch              # Permet de lister les branches
git branch <branche>    # Permet de créer une nouvelle branche <branche>
git branch -m <branche> # Renomme la branche courante en <branche>
git branch -d <branche> # Permet de supprimer une branche
```

- Se placer sur une branche: `git checkout`
- Nouvelle version (git v2.23) : `git switch`

```
git checkout <branche>
git switch <branche>  # Se placer sur la branche <branche>

git checkout -b <branche>
git switch -c <branche>  # Créer la branche <branche> et se positionner dessus
```

FUSIONNER DES BRANCHES

- Le merge permet de ramener une branche sur une autre et ainsi de la fusionner. La fusion de 2 branche se fait toujours à partir de la branche principale.
  - La branche "source" sera affectée en récupérant l'historique de la branche ou un commit de fusion
  - La branche fusionnée ne sera pas affectée
  - l'étape de fusion peut se faire directement dans Gitlab

```
git merge <branche> # Fusionne la branche <branche> avec la branche courante
```

```
$ git switch -b ma_feature
  Switched to branch 'ma_feature'
[ .... Travail sur la branche, successions de commits .... ]

$ git switch develop
  Switched to branch 'develop'

$ git merge ma_feature
  Updating ea1b82a..05e9557
  (Summary of changes)

$ git branch -d ma_feature
  Deleted branch ma_feature (was 05e9557).

$ git push origin develop
```

- En cas de conflit de fusion (mêmes sections de fichier ayant été modifiées différemment dans les deux branches fusionnées), lancez git status pour connaître les fichiers concernés, éditez-les pour régler les problèmes entourés par <<<<<<< et >>>>>>> puis indexer de nouveau ces fichiers pour signifier que le conflit a été résolu.

BONNES PRATIQUES POUR NE PAS SE PERDRE

- Savoir où vous en êtes avec l'état de vos fichiers
  `git status`
- Identifier les différences entre vos fichiers, branches
  `git diff`
- Identifier les contenus des commits
  `git show`, `git log`

GIT DIFF

- Visualisez les modification entre l'index et votre répertoire de travail courant
  - changements non comités
  - `git diff`

```
$ touch monAutreFichier.txt
$ git diff
$ git add monAutreFichier.txt
$ echo "une nouvelle ligne" > monAutreFichier.txt
$ git diff
diff --git a/monAutreFichier.txt b/monAutreFichier.txt
index 0fe633e..10a5e3d 100644
--- a/monAutreFichier.txt
+++ b/monAutreFichier.txt
@@ -1 +1 @@
-ll
+une nouvelle ligne
$ git commit
$ git diff
```

GIT DIFF

- Visualisez les modification entre l'index et votre répertoire de travail courant
  - changements non comités
  - `git diff`

```
$ touch monAutreFichier.txt
$ git diff
$ git add monAutreFichier.txt
$ echo "une nouvelle ligne" > monAutreFichier.txt
$ git diff
diff --git a/monAutreFichier.txt b/monAutreFichier.txt
index 0fe633e..10a5e3d 100644
--- a/monAutreFichier.txt
+++ b/monAutreFichier.txt
@@ -1 +1 @@
-ll
+une nouvelle ligne
$ git commit
$ git diff
```

GIT DIFF

- Visualisez les modification entre l'index et votre répertoire de travail courant
  - changements non comités
  - `git diff`

```
$ touch monAutreFichier.txt
$ git diff
$ git add monAutreFichier.txt
$ echo "une nouvelle ligne" > monAutreFichier.txt
$ git diff
diff --git a/monAutreFichier.txt b/monAutreFichier.txt
index 0fe633e..10a5e3d 100644
--- a/monAutreFichier.txt
+++ b/monAutreFichier.txt
@@ -1 +1 @@
-ll
+une nouvelle ligne
$ git commit
$ git diff
```

GIT DIFF

- Visualisez les modification entre l'index et votre répertoire de travail courant
  - changements non comités
  - `git diff`

```
$ touch monAutreFichier.txt
$ git diff
$ git add monAutreFichier.txt
$ echo "une nouvelle ligne" > monAutreFichier.txt
$ git diff
diff --git a/monAutreFichier.txt b/monAutreFichier.txt
index 0fe633e..10a5e3d 100644
--- a/monAutreFichier.txt
+++ b/monAutreFichier.txt
@@ -1 +1 @@
-ll
+une nouvelle ligne
$ git commit
$ git diff
```

GIT DIFF

- Visualisez les modification entre l'index et votre répertoire de travail courant
  - changements non comités
  - `git diff`

```
$ touch monAutreFichier.txt
$ git diff
$ git add monAutreFichier.txt
$ echo "une nouvelle ligne" > monAutreFichier.txt
$ git diff
diff --git a/monAutreFichier.txt b/monAutreFichier.txt
index 0fe633e..10a5e3d 100644
--- a/monAutreFichier.txt
+++ b/monAutreFichier.txt
@@ -1 +1 @@
-ll
+une nouvelle ligne
$ git commit
$ git diff
```

GIT DIFF

- Visualisez les modification entre l'index et votre répertoire de travail courant
  - changements non comités
  - `git diff`

```
$ touch monAutreFichier.txt
$ git diff
$ git add monAutreFichier.txt
$ echo "une nouvelle ligne" > monAutreFichier.txt
$ git diff
diff --git a/monAutreFichier.txt b/monAutreFichier.txt
index 0fe633e..10a5e3d 100644
--- a/monAutreFichier.txt
+++ b/monAutreFichier.txt
@@ -1 +1 @@
-ll
+une nouvelle ligne
$ git commit
$ git diff
```

GIT DIFF

- Visualisez les modification entre l'index et votre répertoire de travail courant
  - changements non comités
  - `git diff`

```
$ touch monAutreFichier.txt
$ git diff
$ git add monAutreFichier.txt
$ echo "une nouvelle ligne" > monAutreFichier.txt
$ git diff
diff --git a/monAutreFichier.txt b/monAutreFichier.txt
index 0fe633e..10a5e3d 100644
--- a/monAutreFichier.txt
+++ b/monAutreFichier.txt
@@ -1 +1 @@
-ll
+une nouvelle ligne
$ git commit
$ git diff
```

GIT DIFF

- Visualisez les différences entre 2 commits

```
git diff <commit1> <commit1>
```

- Visualisez les différences entre 2 branches

```
git diff <branche1> <branch2>
```

- Visualisez les différences entre la branche locale et distante

```
git diff develop origin/develop
```

GIT LOG

- Consultez l'historique des commits : `git log`

```
$ git log

commit 27329d3afac51fbf2762428e12f2635d1137c549 (origin/master, origin/HEAD,
Author: Administrator <admin@example.com>
Date:   Mon Feb 15 15:52:52 2021 +0000
    Update README.md

commit b362ea7aa65515dc35ff3a93423478b2143e771d
Author: Administrator <admin@example.com>
Date:   Mon Feb 15 15:52:03 2021 +0000
    Initial commit
```

- pour un affichage concis utilisez l'option --oneline

```
$ git log --oneline

27329d3 (origin/master, origin/HEAD, master) Update README.md
b362ea7 Initial commit
```

```
$ git log --all --decorate --oneline --graph
```

GIT SHOW

- pour le détail d'un commit : `git show <commitId>`

```
$ git show 27329d3

commit 27329d3afac51fbf2762428e12f2635d1137c549 (origin/master, origin/HEAD, master)
Author: Administrator <admin@example.com>
Date:   Mon Feb 15 15:52:52 2021 +0000

    Update README.md

diff --git a/README.md b/README.md
index 9ff40b5..047477f 100644
--- a/README.md
+++ b/README.md
@@ -1,2 +1,8 @@
 # Sample GitLab Project

+This sample project shows how a project in GitLab looks for demonstration purposes.
It contains issues, merge requests and Markdown files in many branches,
+named and filled with lorem ipsum.
+You can look around to get an idea how to structure your project and, when done, you can safely delete th
```

GIT EN MODE AVANCÉ

git stash

- Sauvegarder les modifications du working directory dans une zone tampon pour rendre le working directory propre.
- Possibilité de rejouer les modifications stashées sur la branche en cours OU sur une autre branche
- Peut être vu comme une zone de brouillons
- Cette commande peut être utile en cas de développement avec des changements de contextes réguliers

```
#on bloque les modifs dans la zone tampon
$ git stash

# on récupère les modifs stashées
$ git stash apply # garde le stash pour une utilisation future
$ git stash pop # jette le stash

# pour plusieurs stashs
$ git stash list
$ git stash apply stash@{id}

# Pour effacer un stash
$ git stash drop stash@{id}
# pour effacer toute la liste
$ git stash clear
```

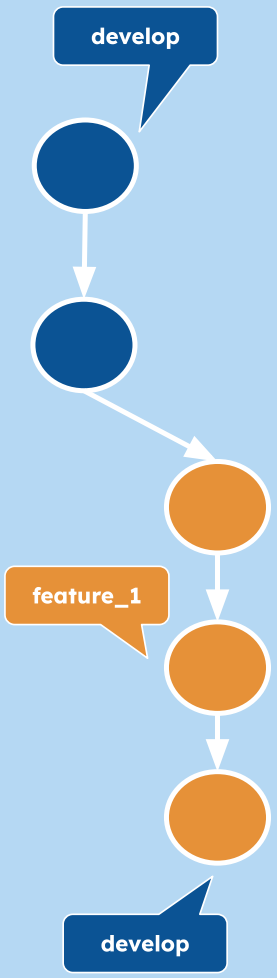
`git merge`

Par défaut et si c'est possible git va opter pour un merge avec *fast-forward*

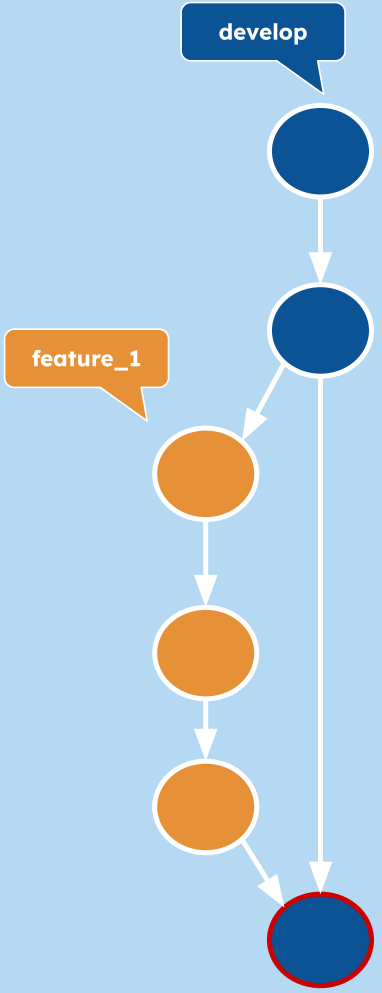
- un chemin linéaire est construit entre les deux branches à fusionner
- les commits de la feature sont fusionnées dans la branche de destination

Sinon si il y a conflit ou si l'option `--no-ff` est utilisée, un commit de fusion est crée en plus des commits de la feature

- avec l'historique de résolution si conflit
- avec un historique des commits de chaque branche



git merge

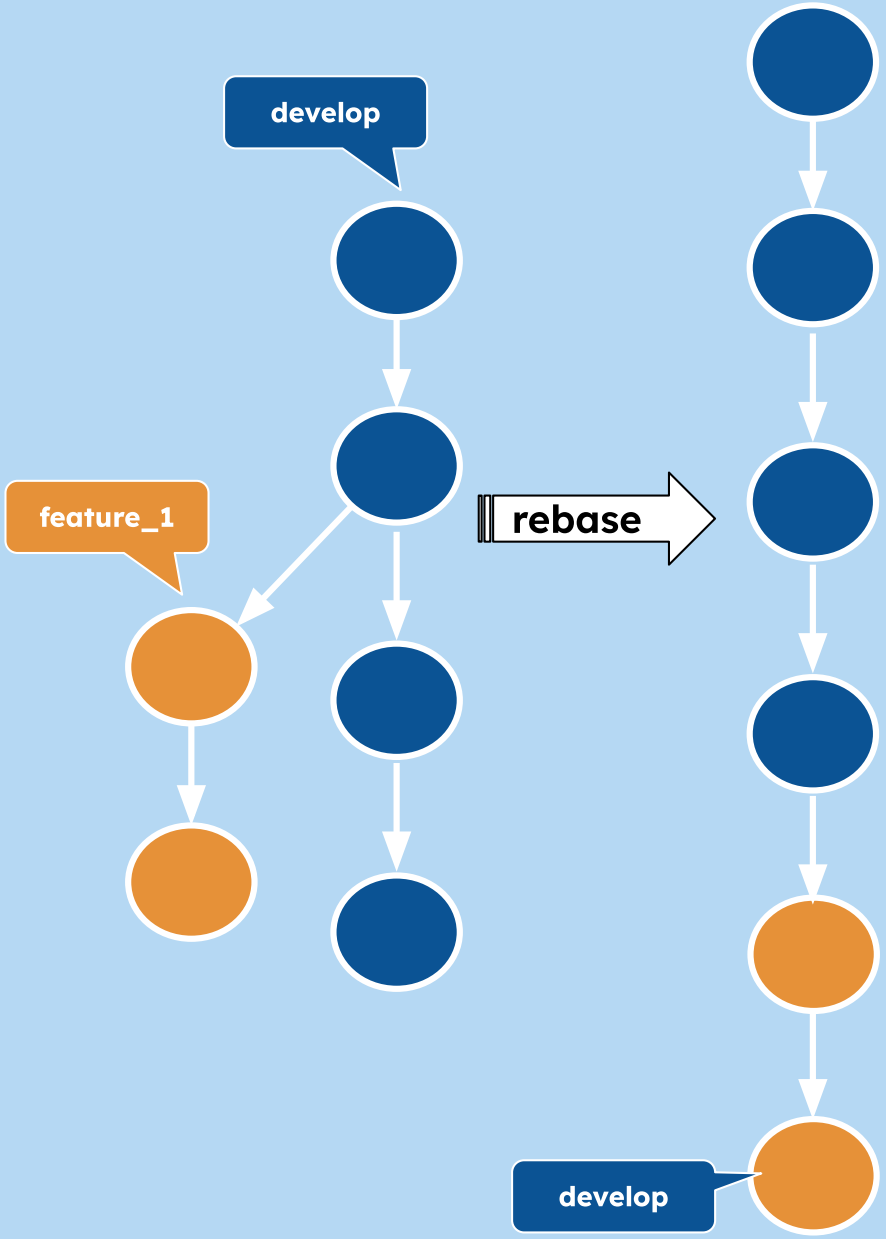


git merge --no-ff

`git rebase`

- Dans le cas où la branche principale a évolué, on ne peut appliquer un `git merge` en mode fast-forward
- On peut en revanche faire un `git rebase`
- Les commits de la branche feature vont être appliqués un par un sur et dans l'ordre sur la branche principale
- On garde un historique linéaire et lisible
- En mode interactif on peut modifier l'ordre

```
# Sur la branche feature
$ git rebase develop
$ git switch develop
$ git merge feature
```


develop
feature_1
rebase
develop

`git merge` VS `git rebase`

- Mettre à jour une branche complètement indépendante, une branche de travail local: `git rebase` ou `git merge` en mode fast forward
  - Évite l'introduction de bruit.
  - Conserve un historique linéaire
- Mettre à jour une branche partagée ou une branche qui risque d'être "conflictuelle": `git merge --no-ff` sans fast-forward.
  - Tous les merges sont représentés par un commit de fusion.
  - Isolation simple des régressions

POUR ALLER PLUS LOIN ...

- La documentation officielle
- Blog de M. Corbin sur git en mode avancé
- Tuto git de Grafikart : manipuler l'historique, le remisage, comment revenir en arrière
- TP et cours git de nos amis de CRIStAL (Centre de Recherche en Informatique, Signal et Automatique de Lille)
- La formation d'Hadrian Grasland
- Les workflows autour de git : Git Flow, le Github Flow, le GitLab Flow