



Formation Gitlab

---

Rappel sur le Git



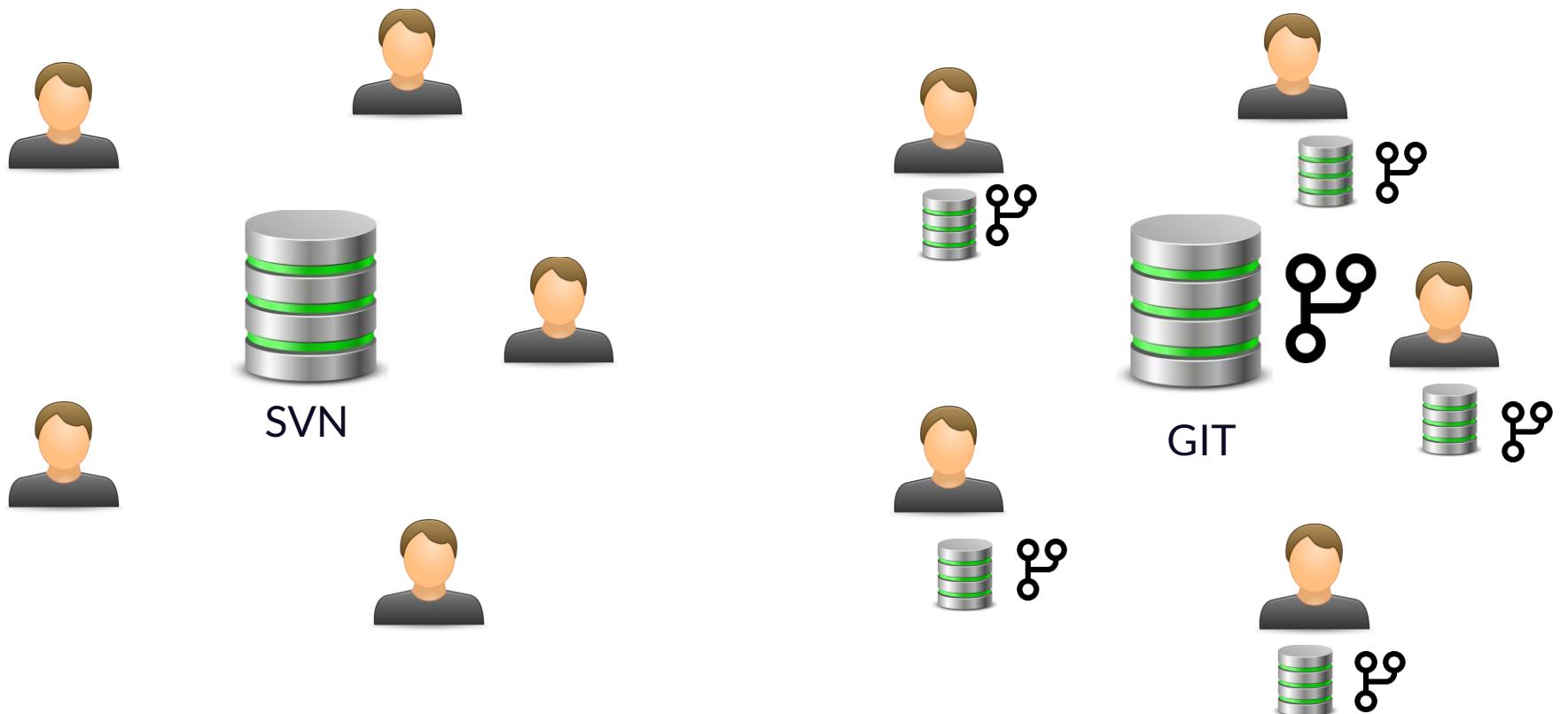
*Béchir BEJAoui*  
Formateur et consultant indépendant

# Rappel sur le Git

1

Introduction

## Git vs SVN:



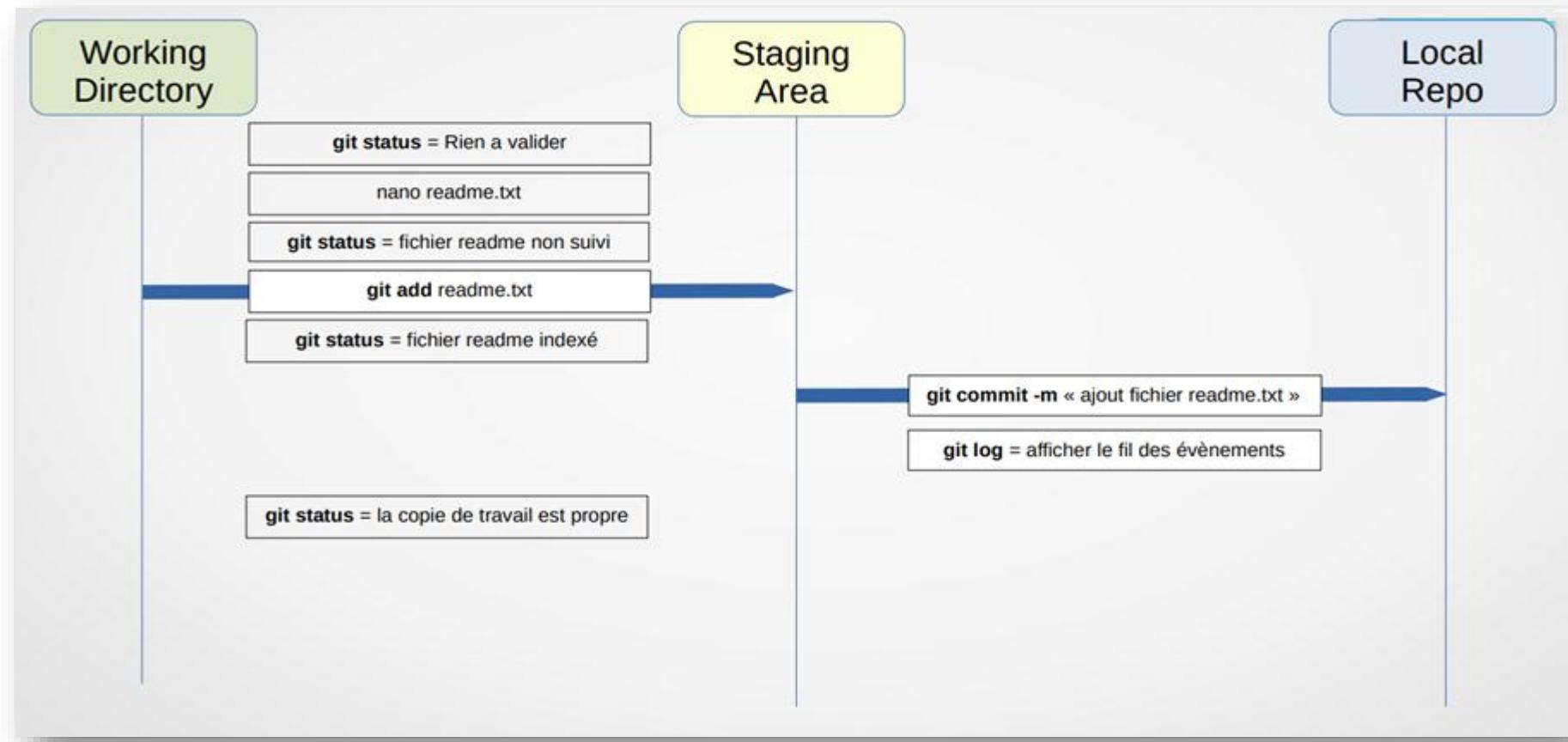
# Rappel sur le Git

1

Introduction

## L'architecture Git :

La commande ci-dessous renvoie une liste d'informations sur votre configuration git, y compris le nom d'utilisateur et l'adresse e-mail :



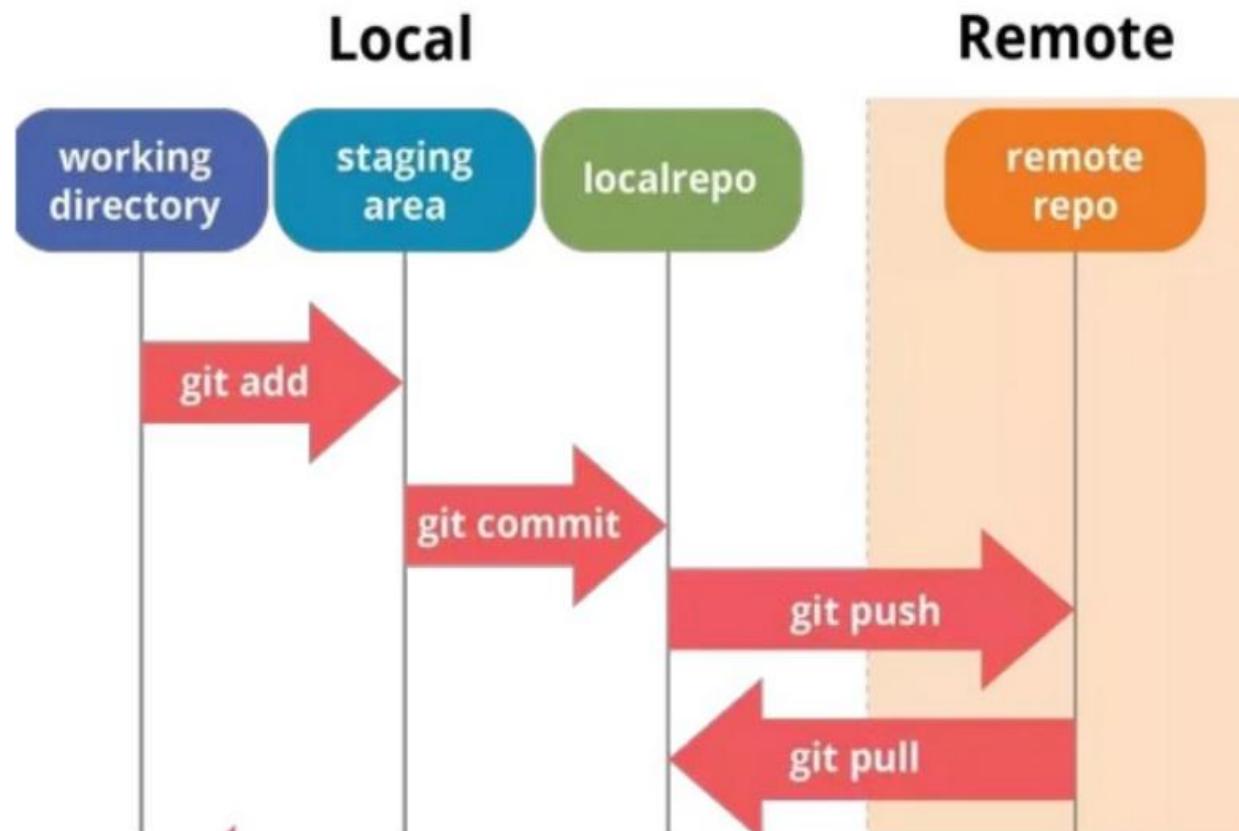
# Rappel sur le Git

1

Introduction

## L'architecture Git :

La commande ci-dessous renvoie une liste d'informations sur votre configuration git, y compris le nom d'utilisateur et l'adresse e-mail :



# Rappel sur le Git

1

Paramètres globaux

## Comment vérifier votre configuration Git :

La commande ci-dessous renvoie une liste d'informations sur votre configuration git, y compris le nom d'utilisateur et l'adresse e-mail :

## Comment configurer votre nom d'utilisateur Git :

Avec la commande ci-dessous, vous pouvez configurer votre nom d'utilisateur :

```
git config --global user.name "Bechir"
```

## Comment configurer votre adresse e-mail utilisateur Git :

Cette commande vous permet de configurer l'adresse e-mail utilisateur que vous utiliserez dans vos commits.

```
git config --global user.email "bechir@xyz.com"
```

# Rappel sur le Git

1

Paramètres globaux

## Comment vérifier votre configuration Git :

La commande ci-dessous renvoie une liste d'informations sur votre configuration git, y compris le nom d'utilisateur et l'adresse e-mail :

**git config -l / git config --list**

```
DELL@PC2023 MINGW64 ~
$ git config list
error: key does not contain a section: li
st

DELL@PC2023 MINGW64 ~
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw
64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
```

1

Paramètres globaux

## Comment mettre en cache vos identifiants de connexion dans Git :

Vous pouvez stocker les informations de connexion dans le cache afin de ne pas avoir à les saisir à chaque fois. Utilisez simplement cette commande :

**git config --global credential.helper cache**

Pour éditer les éléments de la configuration globale tapez

**git config --global --edit**

Ou consultez le fichier .gitconfig sous le dossier d 'utilisateur Windows

Ou /etc/gitconfig sous linux

# Rappel sur le Git

1

Paramètres globaux

2

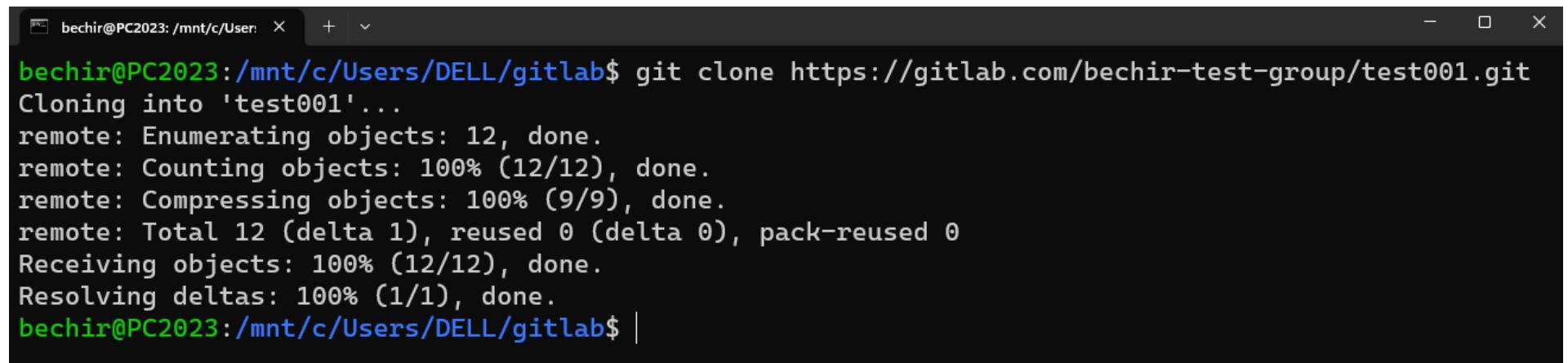
Création du projet

## Création de projet Gitlab (Clonage d'un projet)

Créez un projet Gitlab et puis clonez le dans un répertoire vide là où vous souhaitez placer votre projet avec la commande clone voici un exemple:

**Git clone https://gitlab.com/bechir-test-group/test001.git**

Si le projet contient déjà du code ou du contenu il va être téléchargé, il faut vérifier le dossier



```
bechir@PC2023:/mnt/c/Users/DELL/gitlab$ git clone https://gitlab.com/bechir-test-group/test001.git
Cloning into 'test001'...
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 12 (delta 1), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (12/12), done.
Resolving deltas: 100% (1/1), done.
bechir@PC2023:/mnt/c/Users/DELL/gitlab$ |
```

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

## Création de projet Gitlab (Une première version)

Pour un premier test, ajoutez un fichier en local

```
bechir@PC2023:/mnt/c/Users/DELL/gitlab$ vi main.py|
```

```
bechir@PC2023:/mnt/c/Users/DELL/gitlab$ + v  
print('Hello Python')
```

Lancez la commande **git add main.py**

```
bechir@PC2023:/mnt/c/Users/DELL/gitlab$ git add .|
```

Lancez la commande **git status** et remarquer qu'un fichier est bien ajouté mais il n'est pas encore confirmé avec la commande commit

```
bechir@PC2023:/mnt/c/Users/DELL/gitlab/test001$ git status  
On branch main  
Your branch is up to date with 'origin/main'.  
  
Changes to be committed:  
(use "git restore --staged <file>..." to unstage)  
      new file:   main.py
```

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

## Création de projet Gitlab (Planification)

Enchainez avec la commande **git commit -m message**

**Note:** Le label du commit est arbitraire

```
bechir@PC2023:/mnt/c/Users/DELL/gitlab/test001$ git commit -m "250920231306"
[main 1400de6] 250920231306
 1 file changed, 1 insertion(+)
   create mode 100644 main.py
```

Remarquez que rien n'a changé à distance

Lancez la commande **git push** pour pousser le code vers le répertoire distant

Rafraîchissez la page du projet au niveau de Gitlab et remarquez le nouveau changement

# Rappel sur le Git

1

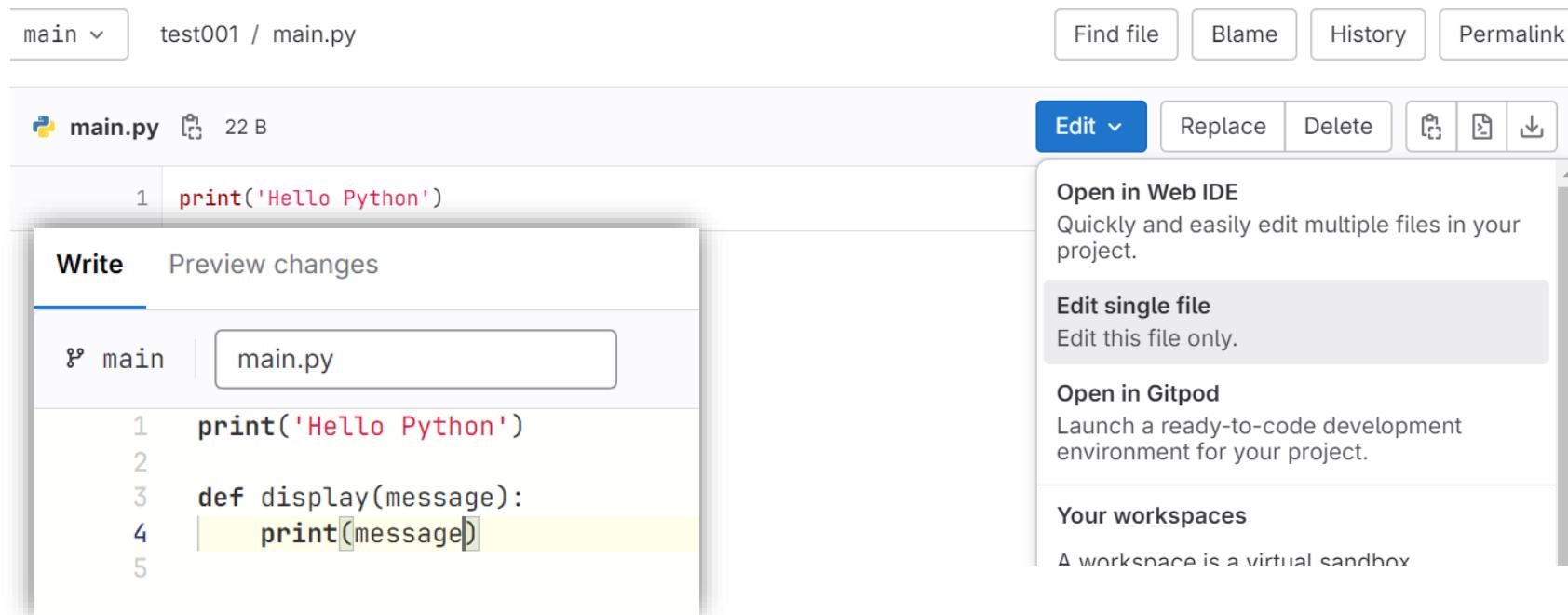
Paramètres globaux

2

Création du projet

## Création de projet Gitlab

Faites un changement à distance pour des fins de test



The screenshot shows a GitLab interface for editing a file named main.py. The file contains the following code:

```
1 print('Hello Python')
```

A modal window titled "Write" is open, showing the same code. The line `print(message)` is highlighted with a yellow background, indicating it has been modified. The "Edit" button is highlighted in blue at the top right of the modal.

At the top of the page, there is a navigation bar with tabs: "main" (selected), "test001", and "main.py". To the right of the tabs are buttons for "Find file", "Blame", "History", and "Permalink". Below the tabs, there are buttons for "Edit", "Replace", "Delete", and icons for "Copy", "Download", and "Raw".

On the right side of the page, there is a sidebar with the following options:

- Open in Web IDE**: Quickly and easily edit multiple files in your project.
- Edit single file**: Edit this file only.
- Open in Gitpod**: Launch a ready-to-code development environment for your project.
- Your workspaces**: A workspace is a virtual sandbox.

Faites un changement au niveau local pour provoquer un conflit

```
bechir@PC2023:/mnt/c/Users/DELL/gitlab/test001$ vi main.py
```



The screenshot shows a terminal window with the following command and output:

```
bechir@PC2023:/mnt/c/Users/DELL/gitlab/test001$ vi main.py
```

```
for x in range(1..10):
    print('Hello Python')
```

The line `for x in range(1..10):` is shown in orange, indicating it is a new or modified line.

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

## Création de projet Gitlab (Planifier un changement)

Lancez un commit avec **git commit -m message** et puis poussez le code vers le répertoire distant avec **git push**

```
bechir@PC2023:/mnt/c/Users/DELL/gitlab/test001$ git push
To https://gitlab.com/bechir-test-group/test001.git
 ! [rejected]      main -> main (fetch first)
error: failed to push some refs to 'https://gitlab.com/bechir-test-group/test001.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.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Remarquez que le conflit de versions locale et distante ne permettent pas la poussée du code vers la branche distante

Lancer la commande **git pull**, il faut constater que la branche locale est synchronisée avec la branche distante

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

## Création de projet Gitlab(La journalisation)

Enfin , lancez la commande **git log** pour vérifier les activités faites

```
bechir@PC2023:/mnt/c/Users/DELL/gitlab/test001$ git log
commit 93ae8400dc5cca0ce4024dfe8deba58c6f02b80c (HEAD -> main, origin/main, origin/HEAD)
Author: Bechir Bejaoui <bejaouibpro@gmail.com>
Date:   Mon Sep 25 11:28:01 2023 +0000

    Update main.py

commit f37854d2fd8e9958c475e177065e5908b25d054b
Author: bechir888 <bejaouibpro@gmail.com>
Date:   Mon Sep 25 13:22:05 2023 +0200

    250920231322

commit 1400de635cdc8d5a1df7d1669c6f39beb6d77a15
Author: bechir888 <bejaouibpro@gmail.com>
Date:   Mon Sep 25 13:07:46 2023 +0200

    250920231306
```

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

## Création de projet Gitlab (La journalisation)

Essayez avec la commande `git log -p` pour ajouter les informations sur les fichiers traités

```
bechir@PC2023:/mnt/c/User: ~ + | v
commit 93ae8400dc5cca0ce4024dfe8deba58c6f02b80c (HEAD -> main, origin/main, origin/HEAD)
Author: Bechir Bejaoui <bejaouibpro@gmail.com>
Date:   Mon Sep 25 11:28:01 2023 +0000

    Update main.py

diff --git a/main.py b/main.py
index f98d9a0..4426b74 100644
--- a/main.py
+++ b/main.py
@@ -1,2 +1,4 @@
-for x in range(1..10):
-    print('Hello Python')
+print('Hello Python')
+
+def display(message):
+    print(message)

commit f37854d2fd8e9958c475e177065e5908b25d054b
Author: bechir888 <bejaouibpro@gmail.com>
Date:   Mon Sep 25 13:22:05 2023 +0200

250920231322
.:|
```

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

## Création de projet Gitlab (La journalisation)

Essayez avec la commande `git diff/git diff --staged` montre les lignes exactes qui ont été ajoutées, modifiées ou effacées

```
$ git diff
diff --git a/fichier.txt b/fichier.txt
deleted file mode 100644
index 76e579a..0000000
--- a/fichier.txt
+++ /dev/null
@@ -1,2 +0,0 @@
-test
-
DELL@PC2023 MINGW64 /c/temp/projects/monprojet (master)
$ git diff --staged
diff --git a/fichier.txt b/fichier.txt
new file mode 100644
index 0000000..76e579a
--- /dev/null
+++ b/fichier.txt
@@ -0,0 +1,2 @@
+test
+
```

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

## Création de projet Gitlab (La journalisation)

Essayez avec la commande `git log -online` permet de montrer un résumé du changement

```
DELL@PC2023 MINGW64 /c/temp/projects/monprojet (master)
$ git log --oneline
7a0da20 (HEAD -> master) content modifié
35ed89b content
```

Pour connaître l'emplacement actuel du HEADER `git show-ref`

La branche actuelle

```
DELL@PC2023 MINGW64 /c/temp/projects/monprojet (master)
$ git show-ref
7a0da20d06c94b48ff3ec0e953584c71bc24aa25 refs/heads/master
```

Le dernier commit

```
$ git diff HEAD~1
diff --git a/content.txt b/content.txt
index f1bdb8b..bf5f0cd 100644
--- a/content.txt
+++ b/content.txt
@@ -1,2 +1,2 @@
-ce si est un texte
+ce si est un texte modifié
```

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

Lisez la sortie de cette façon

```
--- a/main.py
+++ b/main.py
@@ -1,2 +1,4 @@
-for x in range(1..10):
-    print('Hello Python')
+print('Hello Python')
+
+def display(message):
+    print(message)
```

Les ajouts  
4 lignes

Les suppressions  
2 lignes

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

## Création de projet Gitlab (Afficher les détails d'une planification commit)

Pour voir un commit particulier lancez la commande git show suivie par le début de l'identifiant de la commit exemple **git show 7060e**

```
bechir@PC2023:/mnt/c/Users/DELL/gitlab/test001$ git show 7060e
commit 7060e2342ee755359ad6e7c63d34744950f84a92
Author: bechir888 <bejaouibepro@gmail.com>
Date:   Mon Sep 25 12:43:17 2023 +0200

        250920231243

diff --git a/README.md b/README.md
index 65f58a8..4d0dabe 100644
--- a/README.md
+++ b/README.md
@@ -1,92 +1,92 @@
-# test001
-
```

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

## Création de projet (.gitignore)

Pour ignorer certains fichiers dans le répertoire git pour qu'ils ne soient pas pris en considération lors de la pousse vers le répertoire distant, créez un fichier **.gitignore** et validez le

Name	Last commit	Last update
↳ .gitignore	add gitignore file	just now
Ⓜ README.md	250920231243	1 hour ago
🐍 main.py	250920231415	4 minutes ago



Voici quelques règles en relation avec le fichier **.gitignore**

Une ligne commençant par # sert de commentaire

Une ligne commençant par ! sert exception de l'acte d'ignorance

La barre oblique "/" est utilisée comme séparateur de répertoire

Un astérisque "\*" correspond à tout patron

Un point d'interrogation ? correspond à un caractère

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

Voici quelques règles en relation avec le fichier `.gitignore`

- Une ligne commençant par `#` sert de commentaire
- Une ligne commençant par `!` sert exception de l'acte d'ignorance
- La barre oblique `/` est utilisée comme séparateur de répertoire
- Un astérisque `**` correspond à tout patron
- Un point d'interrogation `?` correspond à un caractère
- Deux astérisques `**` correspondent à tout les fichiers et sous dossiers qui se trouvent dans un dossier particulier `abc/**` ou encore à tout sous dossier ou fichier qui correspond à un itinéraire exemple tout dossier ou fichier nommé `file` va être ignoré `*/file`
- Une barre oblique suivie de deux astérisques consécutifs, puis une barre oblique correspond à zéro ou plusieurs répertoires. Par exemple, `"a/**/b"` correspond à `"a/b"`, `"a/x/b"`, `"a/x/y/b"`

.....

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

Voici quelques règles en relation avec le fichier `.gitignore`

- Une ligne commençant par `#` sert de commentaire
- Une ligne commençant par `!` sert exception de l'acte d'ignorance
- La barre oblique `/` est utilisée comme séparateur de répertoire
- Un astérisque `**` correspond à tout patron
- Un point d'interrogation `?` correspond à un caractère
- Deux astérisques `**` correspondent à tout les fichiers et sous dossiers qui se trouvent dans un dossier particulier `abc/**` ou encore à tout sous dossier ou fichier qui correspond à un itinéraire exemple tout dossier ou fichier nommé `file` va être ignoré `*/file`
- Une barre oblique suivie de deux astérisques consécutifs, puis une barre oblique correspond à zéro ou plusieurs répertoires. Par exemple, `"a/**/b"` correspond à `"a/b"`, `"a/x/b"`, `"a/x/y/b"`

.....

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

## Création de projet Gitlab (Annulation de modification non planifiée)

Pour annuler les changements au niveau d'un fichier non encore planifié avec **Commit** utilisez **git checkout nom\_de\_fichier**

Voici un exemple

```
echo "ce ci est un contenu" >> readme
cat readme
git status
git checkout readme
git status
cat readme
```

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

## Création de projet Gitlab (Annulation des planifications)

Il faut comprendre qu'il y a trois niveau d'annulation des planifications

- git reset --soft HEAD~N où N est le nombre de pas d'annulations
- git reset -mixed
- git reset -hard HEAD~N

**HEAD:** est un pointeur ou une référence au dernier commit dans la branche actuelle

**~1:** La dernière planification **Commit** faite

**soft:** Annule la planification des modifications « le Commit »

**mixed (par défaut) :** Annuler la mise en index

**hard:** Annuler la validation + Annuler la mise en index + supprimer les modifications au niveau du fichier, il ne reste plus rien

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

## Création de projet Gitlab (Annulation des planifications à distance)

Il arrive parfois que vous souhaitiez annuler une validation que vous avez poussée vers une branche distante utilisez **git revert** pour annuler les modifications en local et à distance à la fois

Récupérez, tout d'abord, le hachage de validation en utilisant **git reflog**

Puis rétablissez-le. Supposons que mon hachage de validation soit **1255b6910**

Appliquez la commande **git revert 1255b6910**

Si **git revert** échoue

Executer **git cherry-pick --quit HEAD**

Examiner le travail (statut git)

Effectuer des ajustements pour obtenir un statut propre (comme un nouveau commit),  
Refaire **git revert**

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

## Création de projet Gitlab (Récupération en urgence du code détruit)

Au cas de destruction d'un commit en utilisant **--hard**, mais que vous avez ensuite décidé que vous en aviez besoin  
Ne vous inquiétez pas! Il existe encore un moyen pour le récupérer

Tapez **git reflog** vous verrez une liste de SHA de validation. Cette commande affiche un journal des modifications apportées au fichier HEAD. Recherchez maintenant le commit que vous avez détruit et exécutez la commande ci-dessous

```
git checkout -b NewBranchName CommitHashYouDestroyed
```

Vous avez maintenant restauré ce commit. Les commits ne sont pas réellement détruits dans Git avant environ 90 jours, vous pouvez donc généralement revenir en arrière et en sauver un dont vous ne vouliez pas vous débarrasser

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

## Création de projet Gitlab (Récupération montrer les différences)

Pour voir les différences avec les codes précédents déjà ajoutés à l'index

```
bechir@PC2023:/mnt/c/Users/DELL/gitlab/test001$ git reset HEAD -p
diff --git a/README.md b/README.md
index 4d0dabe..8e29087 100644
--- a/README.md
+++ b/README.md
@@ -4,7 +4,7 @@
 
 ## Getting started
-To make it easy for you to get started with GitLab, here's a list of recommended next steps.
+To make it easy for you to get started with , here's a list of recommended next steps.
 
 Already a pro? Just edit this README.md and make it your own. Want to make it easy? [Use the template at the bottom](#editing-this-readme)!

(1/1) Unstage this hunk [y,n,q,a,d,e,?]? y
```

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

## Création de projet Gitlab (Modification du dernier commit)

La commande git commit –amend vous permet de modifier votre dernier commit. Vous pouvez modifier votre message de journal et les fichiers qui apparaissent dans le commit. La solution est la commande **git commit –amend**

```
git commit --amend -m "MAJ dernier commit"
```

Faites les modifications nécessaires et puis enchainez avec

```
git commit --amend --no-edit
```

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

## Création de projet Gitlab (Sauvegarde brouillon)

Vous pouvez ranger les modifications dans un répertoire de travail brouillon avec **git stash**

```
DELL@PC2023 MINGW64 /c/temp/myproject (master)
$ git stash
warning: in the working copy of 'fichier.txt', LF will be replaced by CRLF the next
time Git touches it
Saved working directory and index state WIP on master: 8fe8b4e test

DELL@PC2023 MINGW64 /c/temp/myproject (master)
$ git log
commit 8fe8b4e68236f2cf7c599591e9ec7d29b96dc6d9 (HEAD -> master)
Author: bejaoui bechir <me780411@gmail.com>
Date:   Thu Oct 5 08:14:02 2023 +0200

    test
```

Rien n'a changé

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

## Création de projet Gitlab (Sauvegarde brouillon)

Pour appliquer les changements c'est avec **git stash apply**

```
DELL@PC2023 MINGW64 /c/temp/myproject (master)
$ git stash apply
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   fichier.txt
```

La modification est indexée

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

## Création de projet Gitlab (Sauvegarde brouillon)

Pour lister les changements c'est avec **git stash list**

```
DELL@PC2023 MINGW64 /c/temp/myproject (master)
$ git stash list
stash@{0}: WIP on master: 8fe8b4e test
```

Pour appliquer un brouillon particulier

**git stash apply** suivi par le numéro

Exemple **git stash apply 0**

Pour décrire le stash

**Git stash -m 'description ou titre'**

La modification est indexée

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

## Création de projet Gitlab (Sauvegarde brouillon)

Pour supprimer un stash **git stash drop <numéro>**

```
DELL@PC2023 MINGW64 /c/temp/myproject (master)
$ git stash drop 0
Dropped refs/stash@{0} (1ac83438f7b46324e7c0a3a0b650fa1e30c46eee)
```

La commande **git pop <numéro>**

Permet d'appliquer le brouillon et en même temps dépiler le brouillon de la pile

Suppression du brouillon

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

3

Gestion des branches

## Gestion des branches

Lorsque vous souhaitez utiliser une branche différente déjà existante

`git checkout nom de branche`

Lorsque vous souhaitez créer une nouvelle branche

`git branch nom de branche/ git checkout -b nom de branche`

Lorsque vous voulez lister les branches

`git branch`

Pour afficher le nom de la branche courante

`git branch | grep '*'`

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

3

Gestion des branches

## Gestion des branches

Pour supprimer une branche

`Git branch -d nom de branche`

Pour fusionner deux branches placez vous dans la branche sujet de mise à jour  
En suite exécutez cette commande

`git merge nom de branche contenant la mise à jour`

Lorsque vous voulez lister les branches

`git branch`

Pour afficher le nom de la branche courante

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

3

Gestion des branches

## Gestion des branches

la commande ci-dessus, montre l'évolution de toutes les branches

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

Parfois vous souhaitez abandonner une fusion et recommencer, vous pouvez exécuter la commande suivante

```
Git merge --abort
```

Cette commande ajoute un référentiel distant à votre référentiel local

```
git add remote https://repo_here
```

Pour afficher les référentiels distants utilisez `git remote -v`

```
bechir@PC2023:/mnt/c/Users/DELL/gitlab/test001$ git remote -v
origin  https://gitlab.com/bechir-test-group/test001.git (fetch)
origin  https://gitlab.com/bechir-test-group/test001.git (push)
```

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

3

Gestion des branches

## Gestion des branches

la commande ci-dessus, montre l'évolution de toutes les branches

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

Parfois vous souhaitez abandonner une fusion et recommencer, vous pouvez exécuter la commande suivante

```
Git merge --abort
```

Cette commande ajoute un référentiel distant à votre référentiel local

```
git add remote https://repo_here
```

Pour afficher les référentiels distants utilisez **git remote -v**

```
bechir@PC2023:/mnt/c/Users/DELL/gitlab/test001$ git remote -v
origin  https://gitlab.com/bechir-test-group/test001.git (fetch)
origin  https://gitlab.com/bechir-test-group/test001.git (push)
```

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

3

Gestion des branches

## Gestion des branches

Pour afficher les référentiels distants également utilisez **git remote show origin**

```
bechir@PC2023:/mnt/c/Users/DELL/gitlab/test001$ git remote show origin
Username for 'https://gitlab.com': bechir888
Password for 'https://bechir888@gitlab.com':
* remote origin
  Fetch URL: https://gitlab.com/bechir-test-group/test001.git
  Push URL: https://gitlab.com/bechir-test-group/test001.git
  HEAD branch: main
  Remote branch:
    main tracked
  Local branch configured for 'git pull':
    main merges with remote main
  Local ref configured for 'git push':
    main pushes to main (local out of date)
```

Récupérez les modifications distantes à l'aide de **git fetch** ou **git log origin nom de branche**  
Ou **git remote update**

Fusionnez une branche locale avec une distante

```
git merge origin/main
```

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

3

Gestion des branches

## Gestion des branches

Pour créer une branche distante

`git push -u origin nom de branche`

Pour supprimer une branche distante

`git push --delete origin nom de branche`

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

3

Gestion des branches

## Gestion des branches(Merge vs Rebase)

Merge: Vous êtes sur la branche non encore mise à jour

`git merge nom de branche`

Rebase: La branche 1 confient les modifications

`git rebase nom de branche 1 nom de branche 2`

Il faut surtout éviter les conflits, il faut pas que les deux branches comportent des modifications distinctes sinon au cas de conflit faites

`git merge –abort / git rebase –abort`

Essayez de résoudre les conflits manuellement avant de recommencer l'opération

# Rappel sur le Git

1

Paramètres globaux

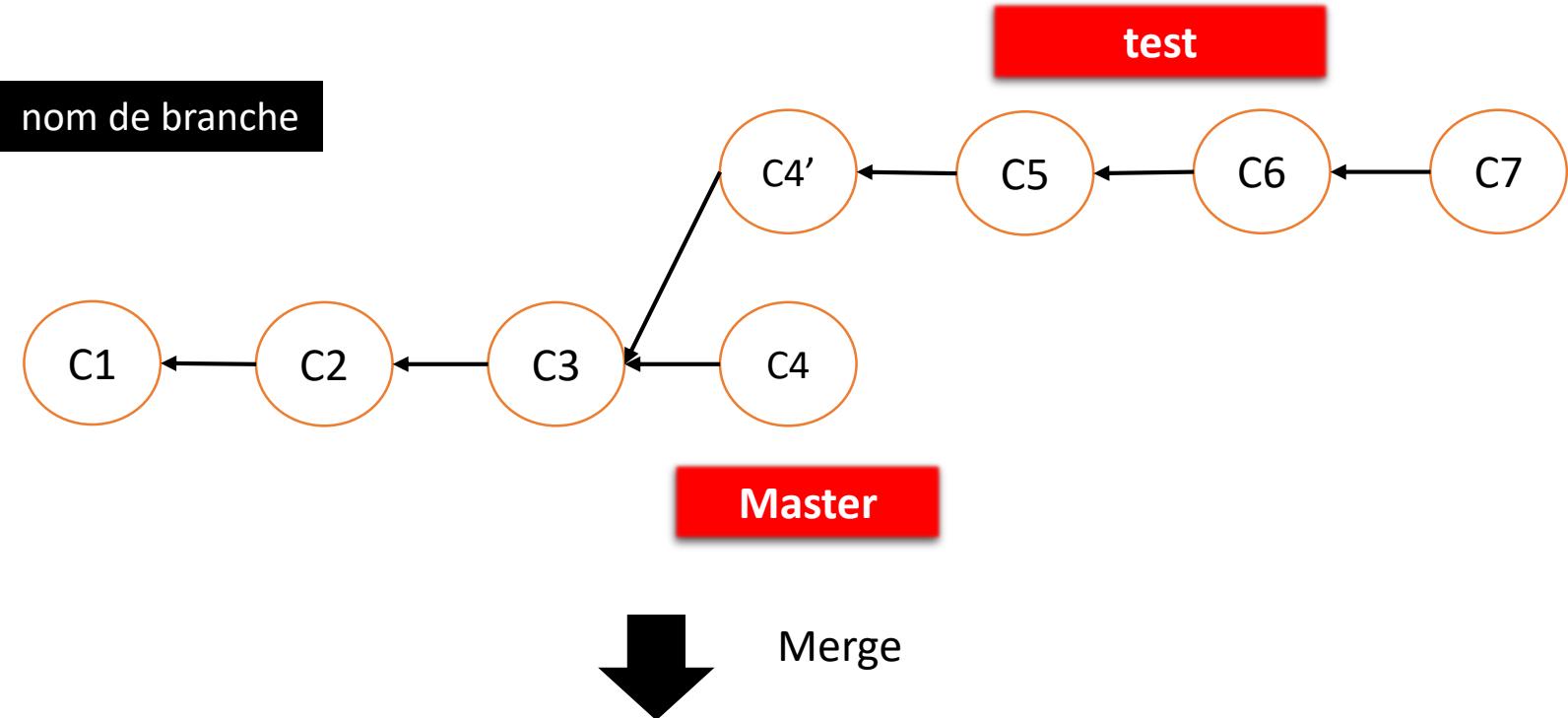
2

Création du projet

3

Gestion des branches

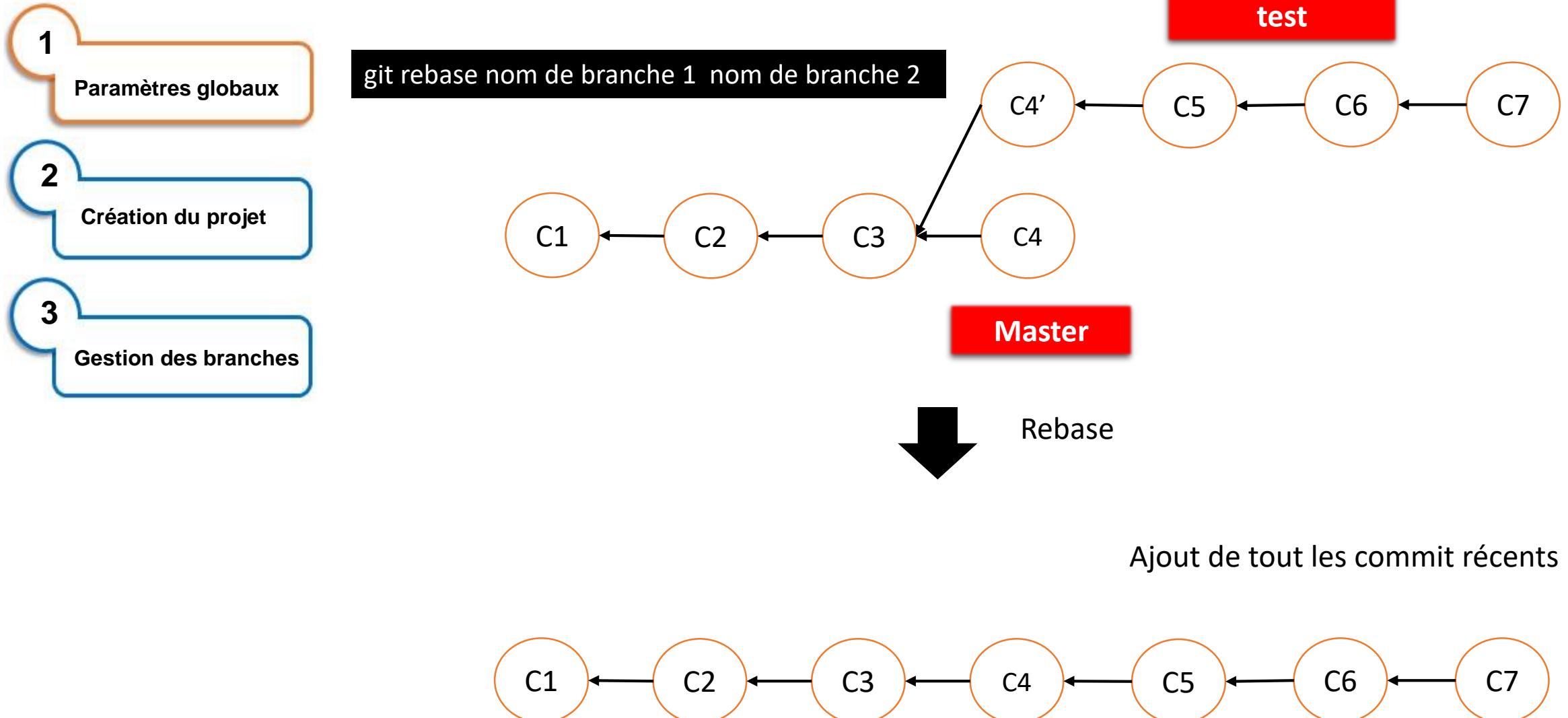
git merge nom de branche



Un commit qui contient toute les modifications



# Rappel sur le Git



# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

3

Gestion des branches

## Gestion des branches (Résolution des conflits avec MergeTool)

Pour résoudre les conflits entre deux branches il y a deux manières

Manuellement

Via un outil de résolution **MergeTool**

Il existe plusieurs MergeTool tierce parties comme

Meld <https://meldmerge.org/>

DiffMerge: <https://sourcegear.com/diffmerge/downloads.php>

Il faut vérifier que la configuration à bien intégré le nouveau outil

**git config -global --list**

```
$ git config --global --list
user.name=bechir888
user.email=bejaouiibpro@gmail.com
user.password=[REDACTED]
credential.http://3.8.22.249.provider=generic
credential.http://ec2-3-8-22-249.eu-west-2.compute.amazonaws.com.provider=generic
diff.tool=diffmerge
difftool.diffmerge.cmd=C:/Program\ Files/SourceGear/Common/DiffMerge/sgdm.exe ""
merge.tool=diffmerge
mergetool.diffmerge.trustexitcode=true
mergetool.diffmerge.cmd=C:/Program\ Files/SourceGear/Common/DiffMerge/sgdm.exe
merge -result="" "" "" "
```

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

3

Gestion des branches

## BRANCHES PROTEGEES:

Dans GitLab, les autorisations sont fondamentalement définies autour de l'idée d'avoir une autorisation de lecture ou d'écriture

Une branche protégée contrôle :

\*Quels utilisateurs peuvent fusionner dans la branche

\*Quels utilisateurs peuvent envoyer vers la branche

\*Si les utilisateurs peuvent forcer le push vers la branche

\*Quels utilisateurs peuvent déprotéger la branche

# Rappel sur le Git

1

Paramètres globaux

2

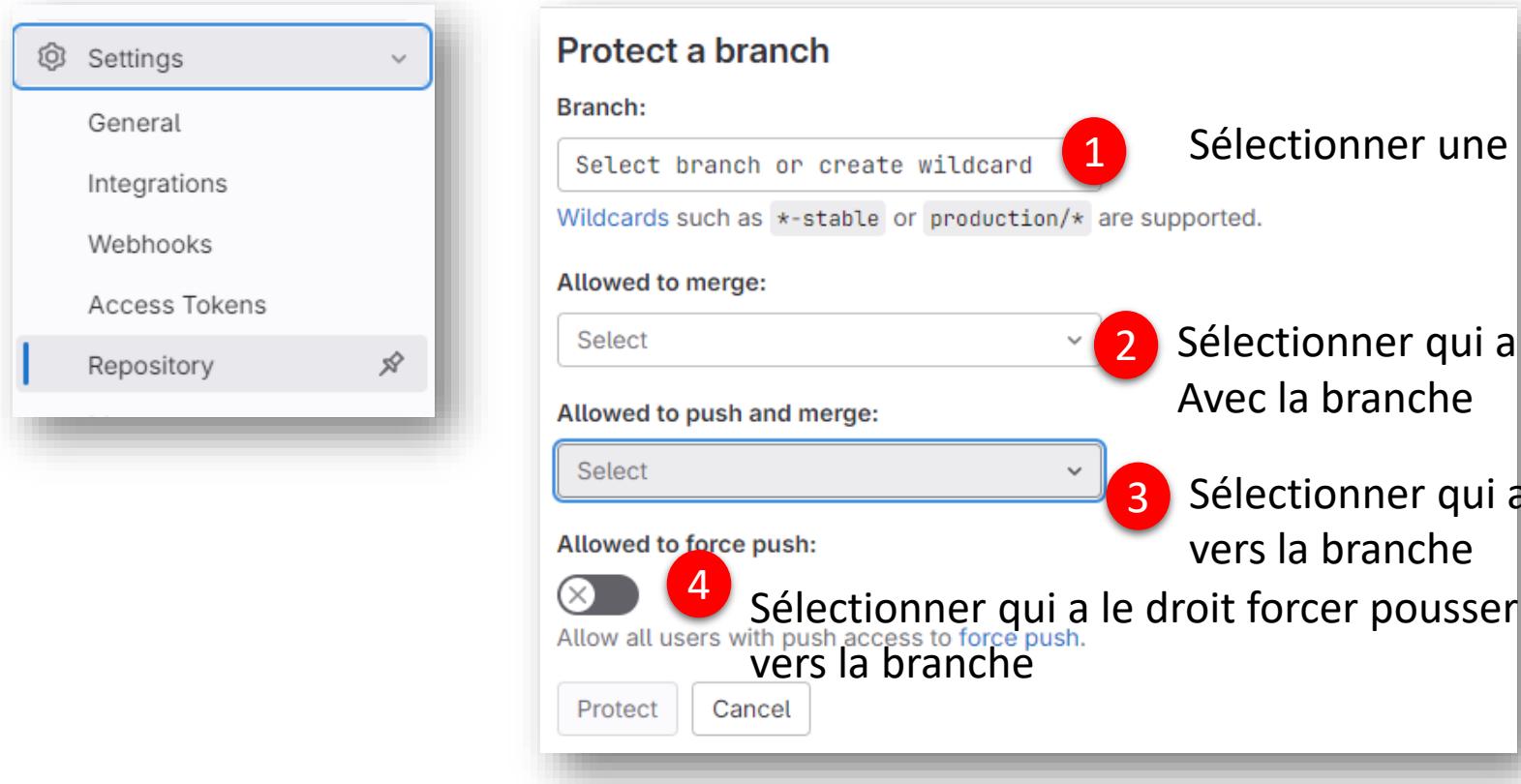
Création du projet

3

Gestion des branches

## BRANCHES PROTEGEES:

Pour créer une branche protégée dans **Paramètres > Répertoires** il faut choisir la section branche protégées



# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

3

Gestion des branches

## LES TAGS:

Dans Git/Gitlab, les tags représentent des points de restauration dans le temps

Il existe deux types de tags

**Les Lightweight:** Ces balises sont généralement créées pour les versions publiques. Les balises annotées contiennent des métadonnées supplémentaires

**Les Annotated:** Ces balises sont utilisées à des fins internes. Ils pointent uniquement vers une version de commit

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

3

Gestion des branches

## LES TAGS:

Pour créer une Tag Annotated:

```
DELL@PC2023 MINGW64 /c/Projets/projet1 (main)
$ git tag -a v1.0.0 4ae883dbcf864ecfa9f53225b04857b99c5486b8 -m "Première version"
```

option -a

Nom

SHA du commit

option -m

Label de tag

Pour créer une Tag Lightweight:

```
DELL@PC2023 MINGW64 /c/Projets/projet1 (main)
$ git tag v1.0.1
```

Pour lister les tags:

```
DELL@PC2023 MINGW64 /c/Projets/projet1 (main)
$ git tag
v1.0.0
```

Pour lister les tags distantes:

```
DELL@PC2023 MINGW64 /c/Projets/projet1 (main)
$ git ls-remote --tags
From https://gitlab.com/bechir888/projet1.git
```

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

3

Gestion des branches

## LES TAGS:

\* Joue le rôle du Jocker

Pour chercher des tags particulières:

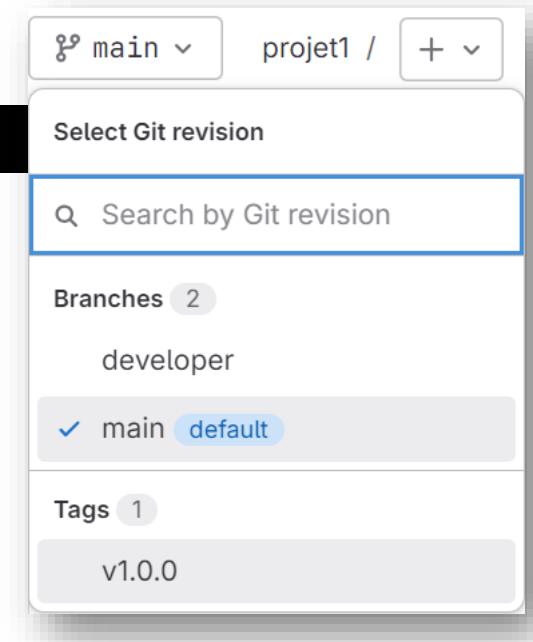
```
DELL@PC2023 MINGW64 /c/Projets/projet1 (main)
$ git tag -l v1*
v1.0.0
```

Pour pousser une tag vers le site distant:

```
DELL@PC2023 MINGW64 /c/Projets/projet1 (main)
$ git push origin v1.0.0
Enumerating objects: 1, done.
Counting objects: 100% (1/1), done.
Writing objects: 100% (1/1), 172 bytes | 172.00 KiB/s, done.
Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
To https://gitlab.com/bechir888/projet1.git
 * [new tag]           v1.0.0 -> v1.0.0
```

Pour pousser toutes les tags:

```
DELL@PC2023 MINGW64 /c/Projets/projet1 (main)
$ git push origin --tags
```



# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

3

Gestion des branches

## LES TAGS:

Il est possible d'effectuer des modifications éloignées des branches à fin de décider d'appliquer ou ne pas appliquer ces modifications

```
DELL@PC2023 MINGW64 /c/Projets/projet1 (main)
$ git checkout v1.0.0
Note: switching to 'v1.0.0'.
```

Vous êtes actuellement en mode détaché

```
DELL@PC2023 MINGW64 /c/Projets/projet1 ((v1.0.0))
```

```
$ git checkout v1.0.0
Note: switching to 'v1.0.0'.
```

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using -c with the switch command. Example:

```
git switch -c <new-branch-name>
```

Or undo this operation with:

```
git switch -
```

Vérification du pointeur avec la commande **git branch**

```
DELL@PC2023 MINGW64 /c/Projets/projet1 ((v1.0.0))
$ git branch
* (HEAD detached at v1.0.0)
  developer
  main
```

Le pointeur est en mode détaché les modifications ne sont pas appliquées

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

3

Gestion des branches

## LES TAGS:

Pour supprimer une tag locale

```
DELL@PC2023 MINGW64 /c/Projets/projet1 ((v1.0.0))
$ git tag -d v1.0.0|
```

Pour supprimer une tag distante

```
DELL@PC2023 MINGW64 /c/Projets/projet1 ((v1.0.0))
$ git push --delete origin v1.0.0|
```

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

3

Gestion des branches

## LE MODEL GITFLOW

Le model git flow est un arrangement qui permet d'organiser la création des branches au sein d'un projet pour éviter les conflits pendant les demandes de fusions

Les règles sont claires :

- La branche main est la branche principale à ne pas toucher qu'à la dernière minute en phase de production ou lors de correction de Bug sévère niveau production
- La branche Develop est dérivée de la branche main elle servira comme branche de base pour les branches features temporaires
- Vous ajouterez des branches **features** pour les versions *majeures et mineures*,
- Vous créerez des **release** pour des versions *mineures* uniquement
- Vous créerez des **hotfix** des bugs pour les versions *patch* au niveau production

# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

3

Gestion des branches

## LE MODEL GITFLOW

Le model git flow est un arrangement qui permet d'organiser la création des branches au sein d'un projet pour éviter les conflits pendant les demandes de fusions

Les règles sont claires :

- La branche main est la branche principale à ne pas toucher qu'à la dernière minute en phase de production ou lors de correction de Bug sévère niveau production
- La branche Develop est dérivée de la branche main elle servira comme branche de base pour les branches features temporaires
- Vous ajouterez des branches **features** pour les versions *majeures et mineures*,
- Vous créerez des **release** pour des versions *mineures* uniquement
- Vous créerez des **hotfix** des bugs pour les versions *patch* au niveau production

# Rappel sur le Git

1

Paramètres globaux

2

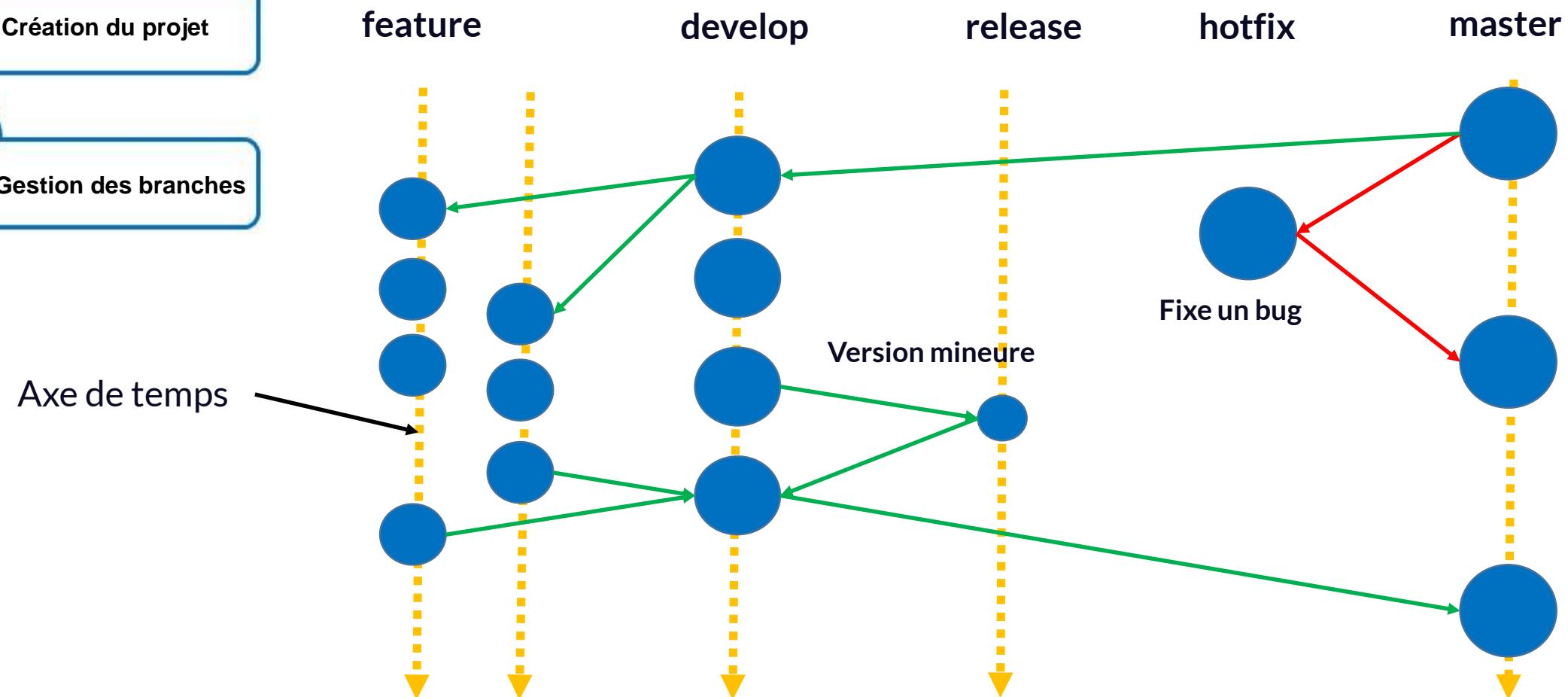
Création du projet

3

Gestion des branches

## LE MODEL GITFLOW

Le git flow peut être implémenté manuellement ou via le module Gitflow de git



# Rappel sur le Git

1

Paramètres globaux

2

Création du projet

3

Gestion des branches

## LE MODEL GITFLOW

Le git flow peut être implémenté manuellement ou via le module Gitflow de git

