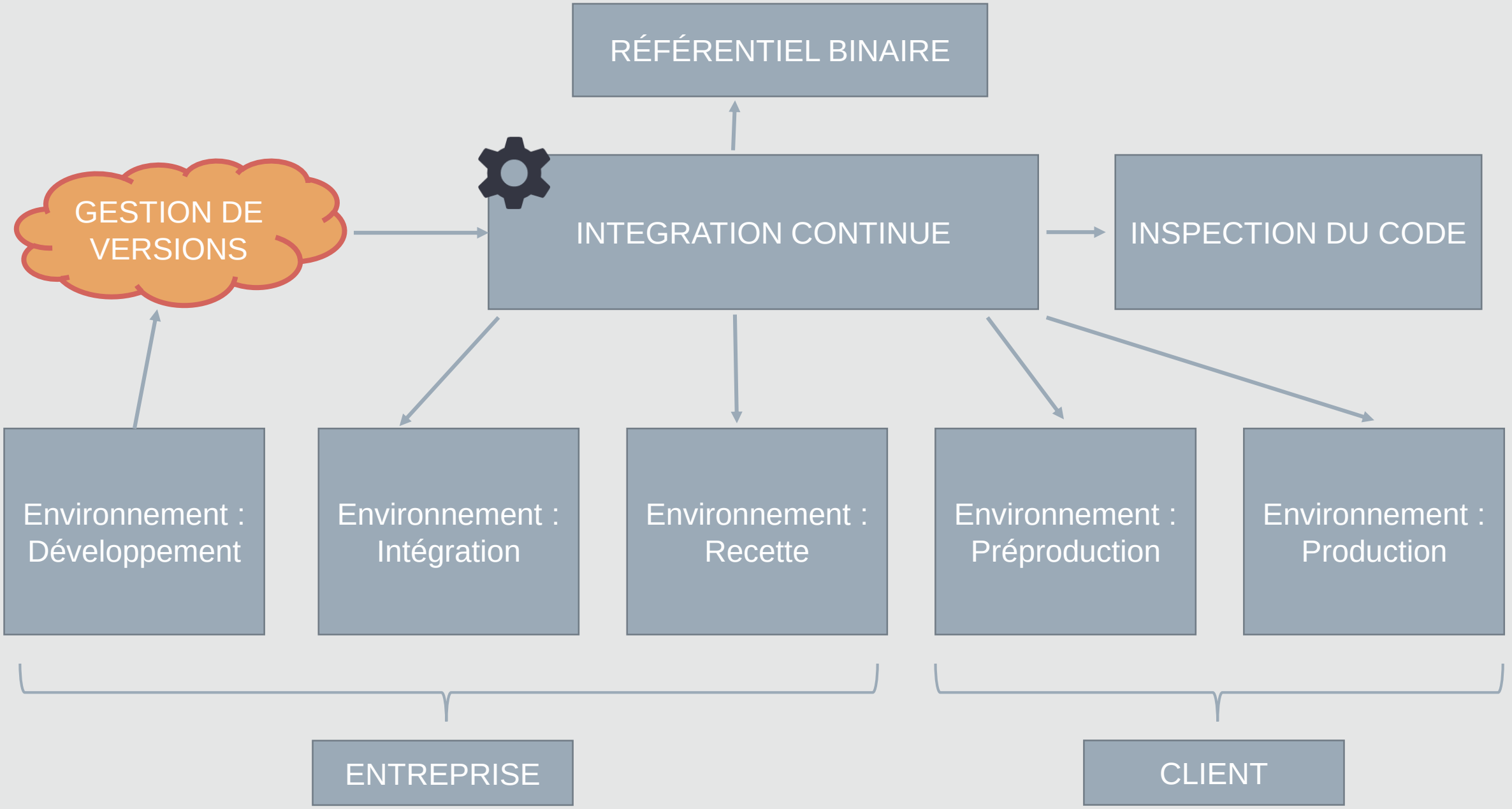
USINE LOGICIELLE

01/2019 | GAËL SIGOGNEAU | V1.0

M02 : GESTION DE VERSION

SITUER LE GESTIONNAIRE DE VERSION



Plan du module :

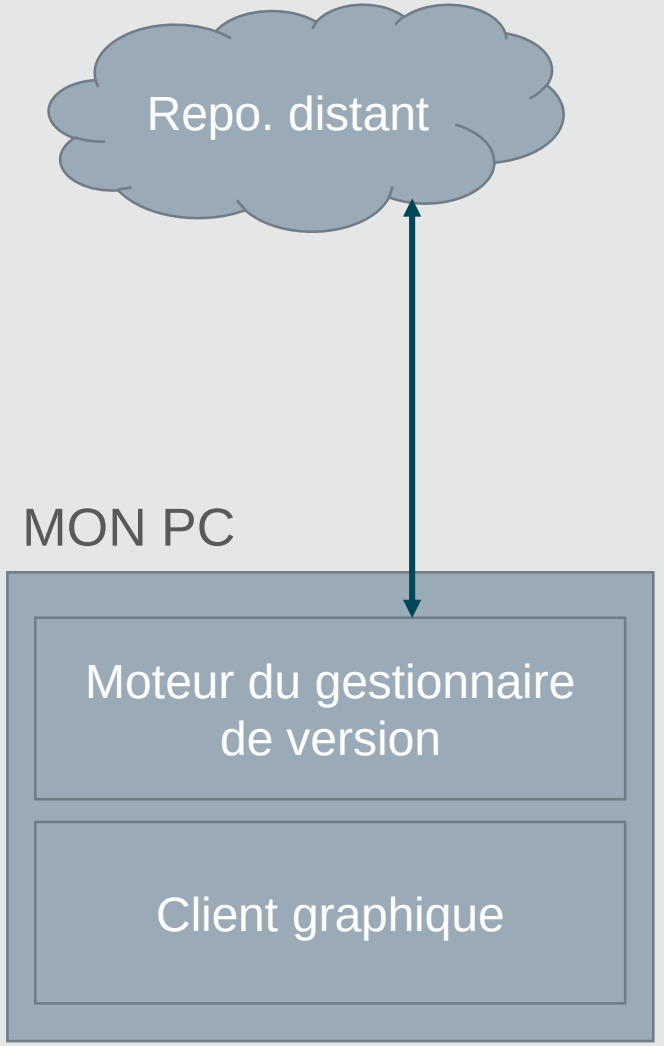
- LES GESTIONNAIRES DE VERSION
- GIT :
  - Fonctionnement
  - Les principales commandes SHELL
  - Les fichiers de configuration GIT
  - Organisation d’un repo. : les branches
  - La gestion des conflits
  - GIT-FLOW

Les 2 principaux gestionnaire de version :

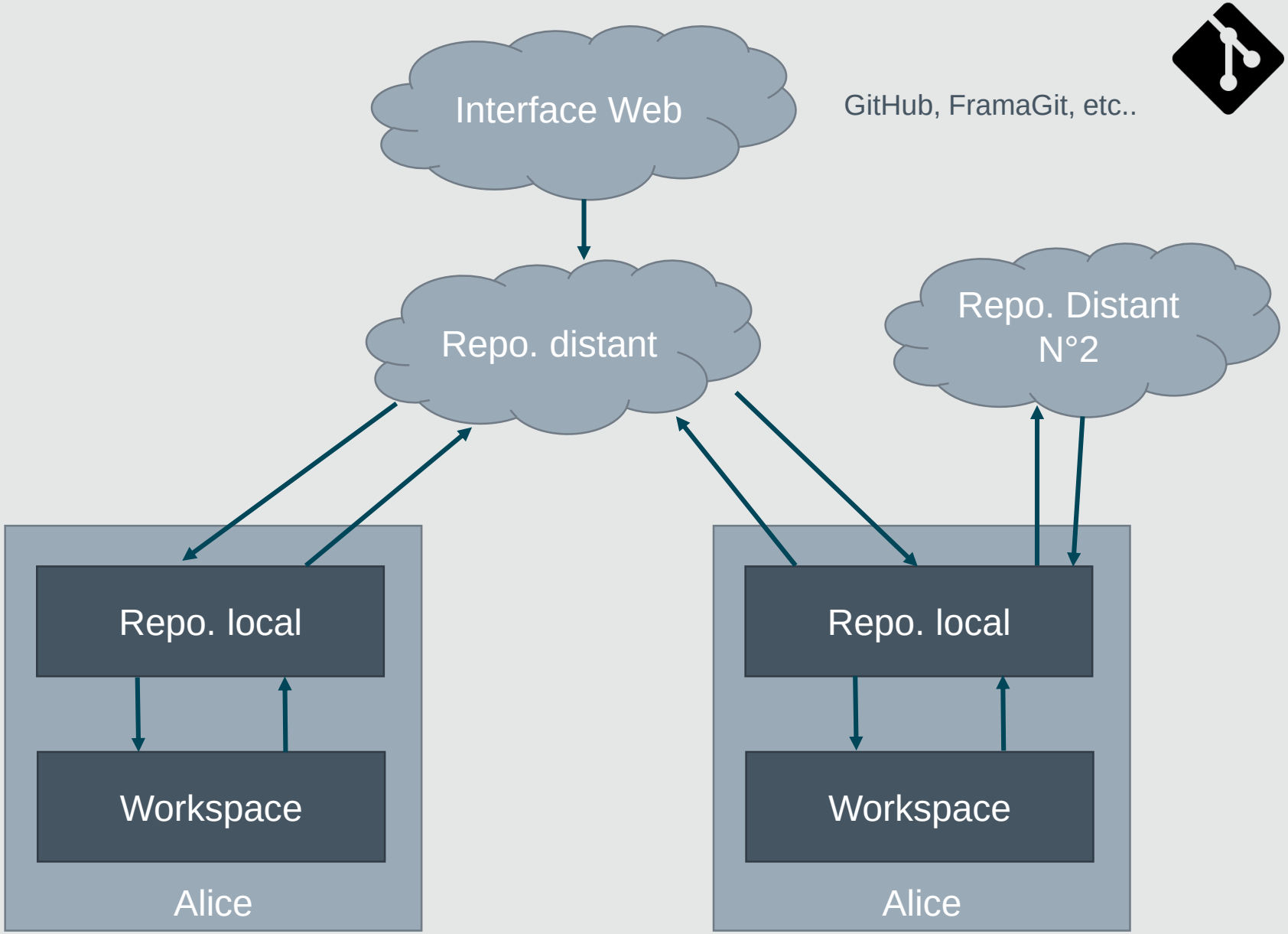
- **SVN** (vieillissant)
- **GIT** (ne pas confondre avec GitHub, etc..)

Fonctionnement d'un gestionnaire de version :

- Le moteur du gestionnaire de version est la brique applicative chargée de versionner le code
- Le client graphique est une coquille vide. D'ailleurs, il y a plusieurs clients graphiques pour chaque moteur de versionning



Fonctionnement de GIT :



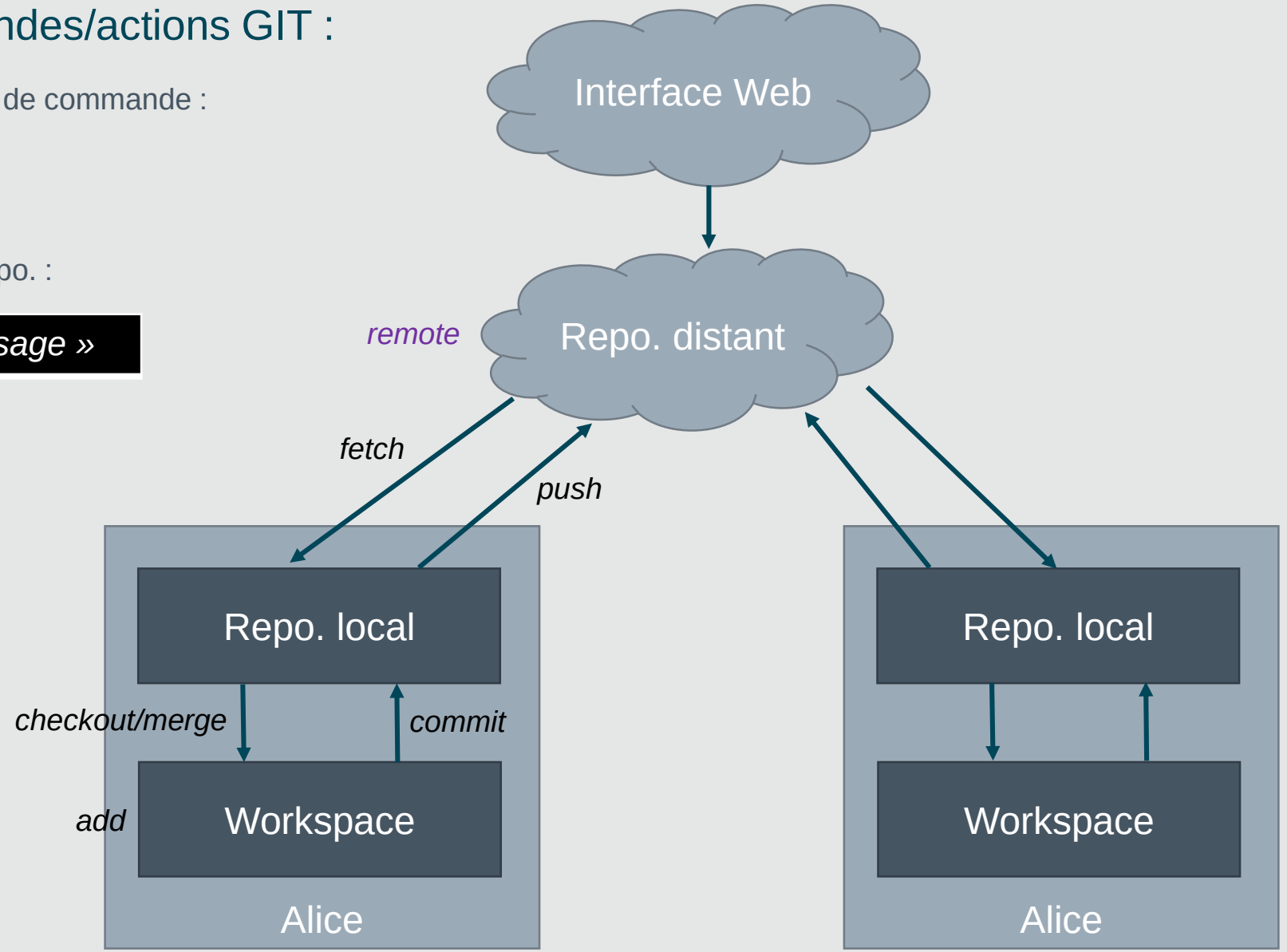
Les principales commandes/actions GIT :

Syntaxe pour utiliser GIT en ligne de commande :

```
> git […]
```

Exemple : Envoyer du code au repo. :

```
> git commit –m « mon message »
```



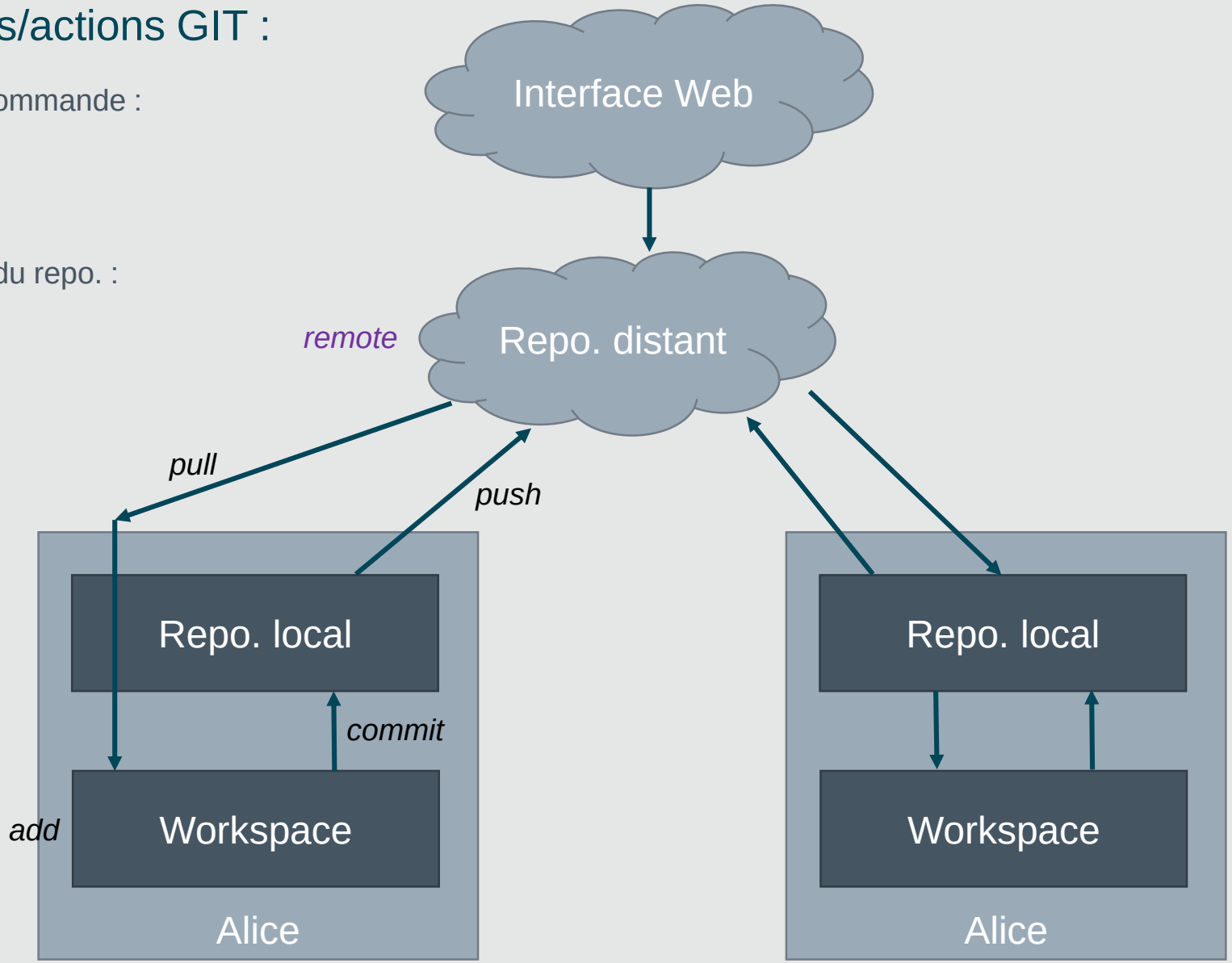
Les principales commandes/actions GIT :

Syntaxe pour utiliser GIT en ligne de commande :

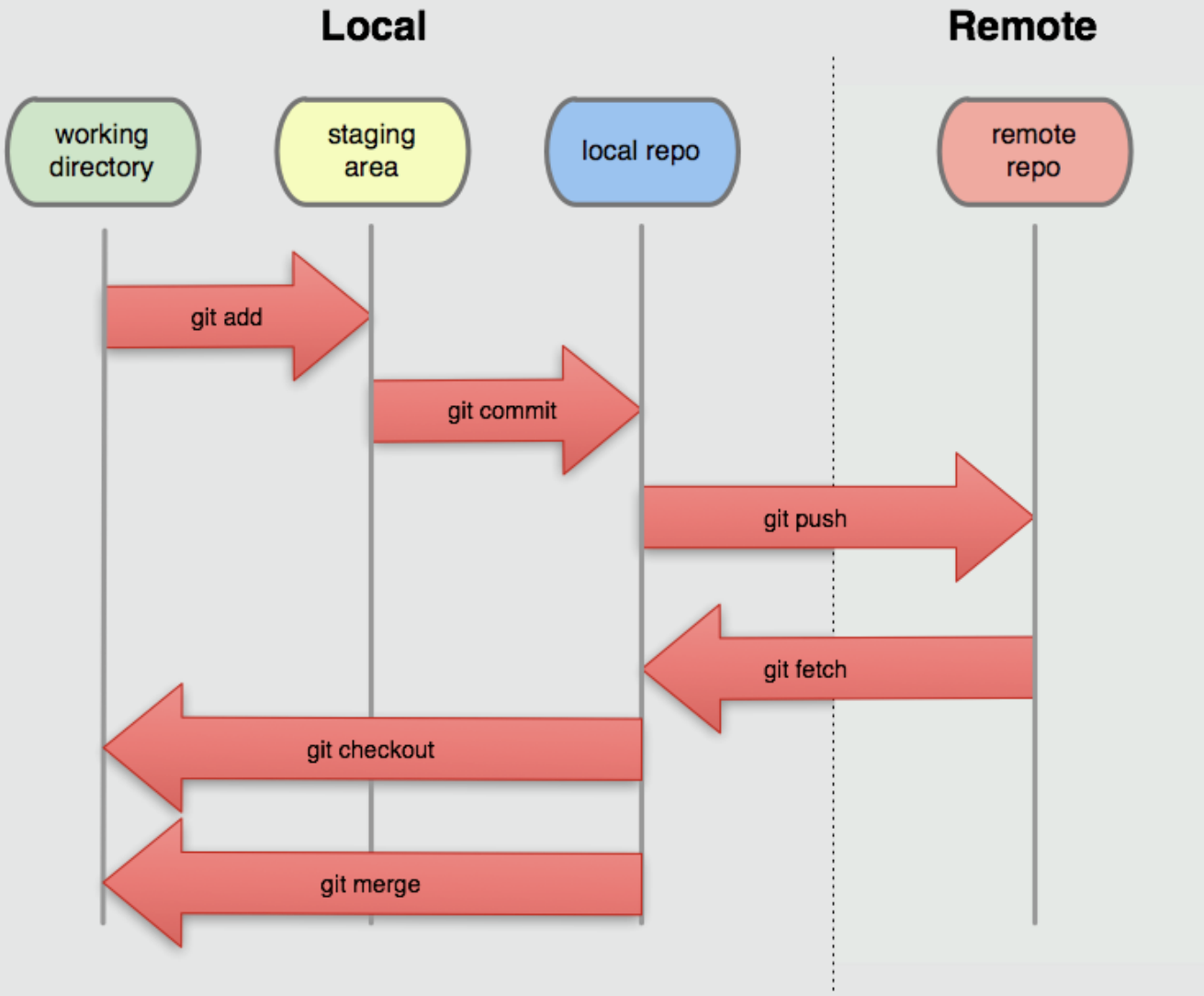
```
> git […]
```

Exemple : récupérer les modifications du repo. :

```
> git pull
```



Les principales commandes/actions GIT :



PREREQUIS

- Installer :
  1. CMDER (Répertoire C:/progiciels)
  2. GitKraken
  3. Tortoise GIT

- Créer un répertoire C:/cours
  - Créer un répertoire pour vos tests : C:/cours/automatisation-demo
  - Réaliser un clone dans le répertoire C:/cours
    - › git clone https://framagit.org/gantispam/automatisation-git.git

En Unix :

```
> mkdir –p /opt/cours/automatisation-demo && cd /opt/cours/
> git clone https://framagit.org/gantispam/automatisation-git.git
```

Les principales commandes/actions GIT :

Commande	Actions
git init	Créer un nouveau repo. local GIT
git add […]	Ajouter un nouveau fichier à versionner dans GIT
git commit […]	Versionner le/les fichiers
git push […]	Envoyer le code et le versionning dans le repo. distant
git fetch […]	Récupérer les modifications du repo. distant sans les appliquer
git checkout […]	Gestion des versions (commit, branche)
git merge […]	Récupérer du code d'une branche à une autre
git pull […]	Récupérer les modifications du repo. distant et les appliquer (git fetch réalisé automatiquement)

Commande	Actions
git clone […]	Copier et synchroniser un repo. distant (git init+remote réalisé automatiquement)

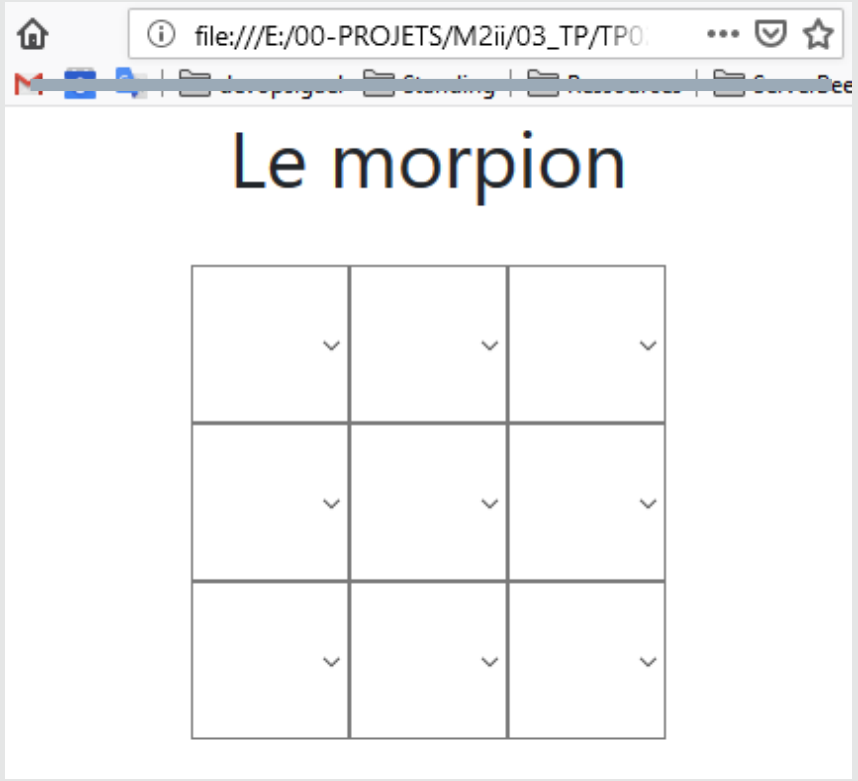
Les principales commandes/gestion GIT :

Commande	Actions
git log	Afficher l'historique des commits
git status	Afficher le statuts des fichiers du repo.
git remote […]	Afficher/Gestion des repo. distants
git branch	Afficher les branches
[…]	[…]

TP

TP01_01-morpion

Travailler en collaboration : le jeu du Morpion



1-Créer le morpion en pair-programming

2-Déponser le morpion sur un repo. distant

Les principaux fichiers spécifiques à GIT :

Fichier	Rôle
README.md	Fichier pour réaliser une « petite » documentation/présentation du code. README.md peut être placé partout dans le système de répertoire du code source. Le fichier est généralement lu et affiché par les clients WEB pour présenter le projet. Ce fichier utilise la syntaxe Markdown pour la mise en forme
.gitignore	Certains fichiers ne doivent pas être versionnés ni envoyés au repo. GIT (exemple : fichier de config des mots de passes, fichiers de config. IDE, etc.) .gitignore permet de lister les répertoires et fichiers à ne pas versionner
.gitkeep	GIT n'est capable de versionner que les fichiers. Dans certain cas la présence de répertoires vides est nécessaire au projet. La création d'un fichier vide nommé .gitkeep permet de forcer le versionning d'un répertoire. Ce fichier doit être placé dans le répertoire à versionner.
.gitmodules	Fichier pour lier un projet à des sous-modules (sous repo. Git)

README.md → La synthaxe Markdown:

Markdown est un langage de balisage léger. Il permet de réaliser une mise en forme très rapidement :

Syntaxe	Rôle
# mon titre H1	Permet de faire un titre de type H1 (c.f HTML)
## titre H2	Permet de faire un titre de type H2 (c.f HTML) Etc. etc.
* Texte italique *	Formater le texte en italique
** Texte gras **	Formater le texte en gras
` du code `	Afficher le texte comme du code source (note : le caractère ` est une apostrophe inversée, par défaut "AltGr + 7" sur les claviers AZERTY)
``java String az = new String(); ```	Bénéficier de la coloration syntaxique d'un langage de programmation
> Une citation	Faire une citation
* Item 1 * Item 2	Faire des listes à puces
1. Item 1 2. Item 2	Faire une liste ordonnée
[texte du lien](url_du_lien "texte pour le titre, facultatif")	Faire un lien hypertexte
![Texte alternatif](url_de_l'image "texte pour le titre, facultatif")	Insérer une image

README.md → La synthaxe Markdown:

Markdown est un langage de balisage léger. Il permet de réaliser une mise en forme très rapidement :

Syntaxe	Rôle
| Titre 1 | Titre 2 | Titre 3 | | :---------- | :----------: | ----------: | | Colonne | Colonne | Colonne | | Alignée à | Alignée au | Alignée à | | Gauche | Centre | Droite |	Faire un tableau en markdown
---	Faire une barre de séparation (ne pas abuser)
	Etc. etc.

Généralement :

- un fichier README.md est présent à la racine du proket
- Plusieurs READM.md possible
- Les balises HTML sont aussi acceptées

.gitignore → Exemple :

README.md

```
# JAVA JEE

## PROGRAMME :

### M01 – Présentation Java :
* Niveau de difficulté : +
* Contenu :
    - présentation fonctionnement
    - Les collections
    - Les streams
    - Heritage
    - Interface
    - Generic
    - JDBC JAVA

### M02 - GRADLE (Moteur de production) :
* Niveau de difficulté : +
* Contenu :
    - Installations
    - Fonctionnement
```

.gitignore → La syntaxe :

Le fichier .gitignore permet de ne pas versionner certains fichiers et répertoires :

Syntaxe	Rôle
# un commentaire	Faire un commentaire
Monfichier.txt	Ignorer récursivement les fichiers portant le nom « MonFichier.txt »
monRepertoire/	Ignorer les répertoires portant le nom « monRepertoire »
*.doc	Ignorer les fichiers avec l'extension « .doc »
*_demo.java	Ignorer les classes java finissant par « _demo »
!finalementNon.doc	Ne pas ignorer les fichiers Word portant le nom « finalementNon.doc »

Généralement :

- un fichier .gitignore est présent à la racine du projet.
- Le fichier .gitignore peut être surchargé dans les sous répertoires

.gitignore → Exemple :

.gitIgnore

```
# IDE
.vscode/
.idea/
.settings/
bin/
.classpath
.project
out/

# GRADLE
.gradle
build/
gradle/
.gradletasknamecache
!gradle-wrapper.jar
gradlew
gradlew.bat

# Maven
target/
.mvn/
```

.gitmodules → Exemple :

.gitmodules

```
[submodule "shFlags"]
        path = shFlags
        url = git://github.com/nvie/shFlags.git
```

Les concepts de GIT :

Les branches

- GIT est basé sur un système de branches, l'utilisateur est libre de s'organiser comme il le souhaite. Généralement :
  - La branche « master » → code mutualisé entre tous les développeurs à l'instant T
  - Les autres branches → code déviant de la version principale (codes en cours, fonctionnalités dérivées, etc.)

Les tags

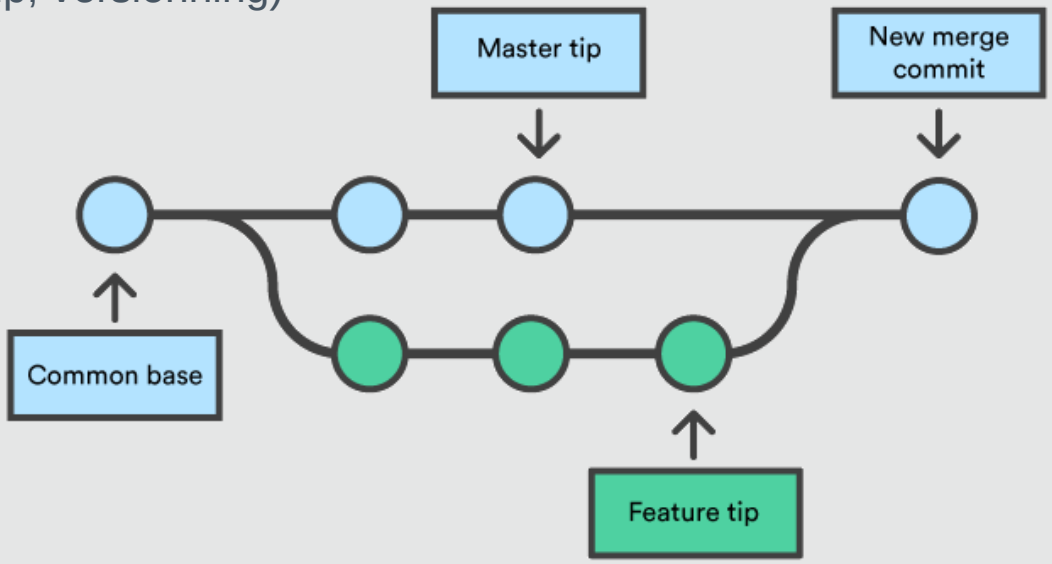
- « Photo instantanée » à un instant T du code Source (backup, versionning)

Les publications

- Versions packagés de l'application

Les PullRequests :

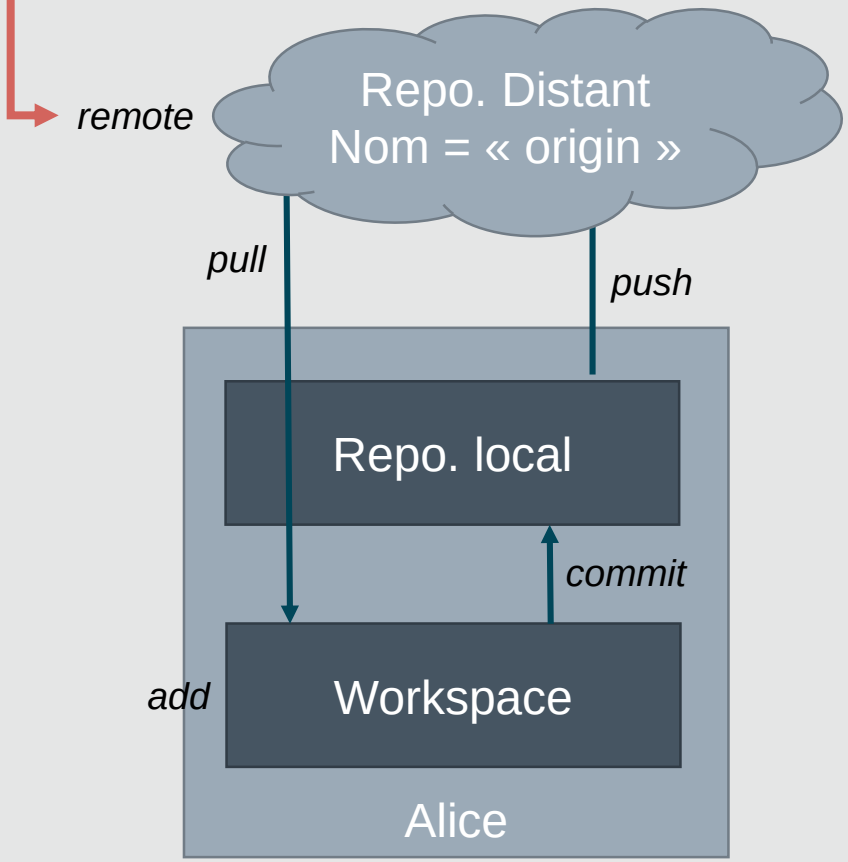
- Proposer du code au développeur référent d'un projet

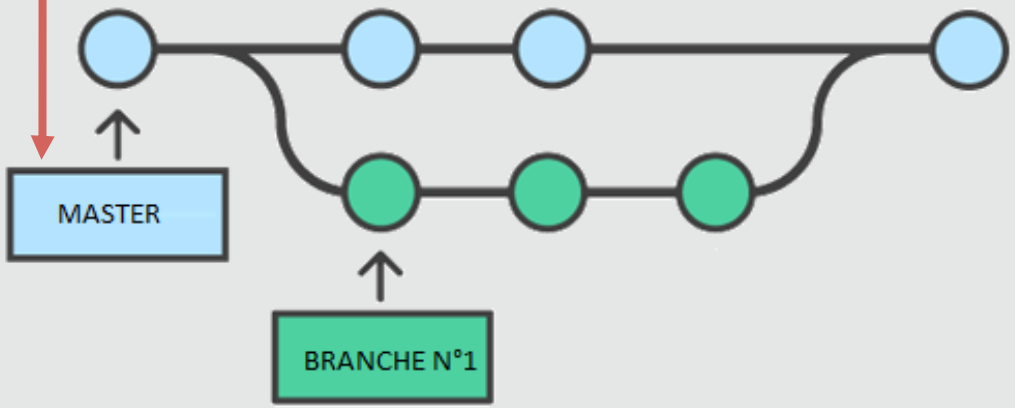


Ne pas confondre REMOTE et BRANCH !

Exemple : récupérer les modifications du repo. :

```
> git pull origin master
```





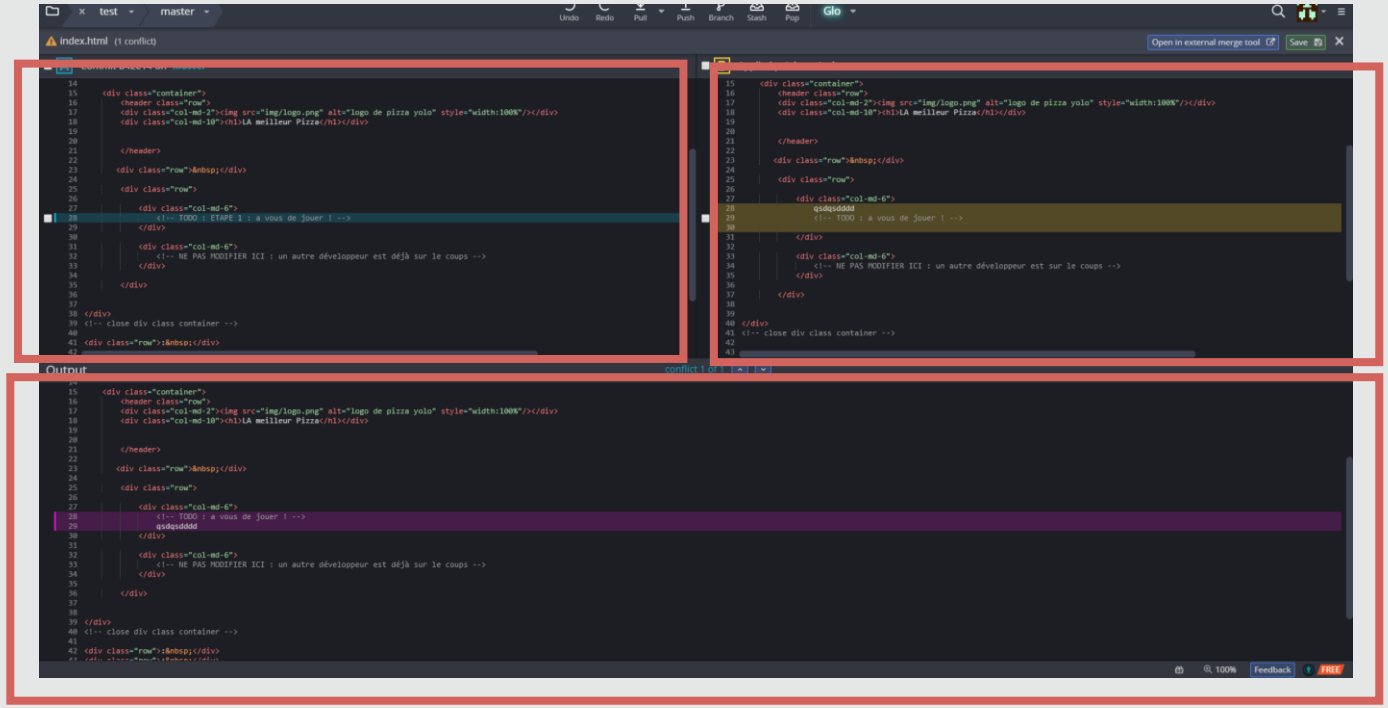
La gestion des conflits

Un travail collaboratif sur un code source amène parfois des développeurs à modifier les mêmes fichiers.
Lors de la fusion du code, des conflits peuvent survenir…

Conseil pour éviter les erreurs de fusion du code :

- Dans le cas où 2 développeurs travaillent sur la même branche : réaliser TOUJOURS un pull avant de faire un PUSH !
- Utiliser un client graphique pour résoudre les conflits

Code récupéré depuis le repo.



Votre code

Version du code finale

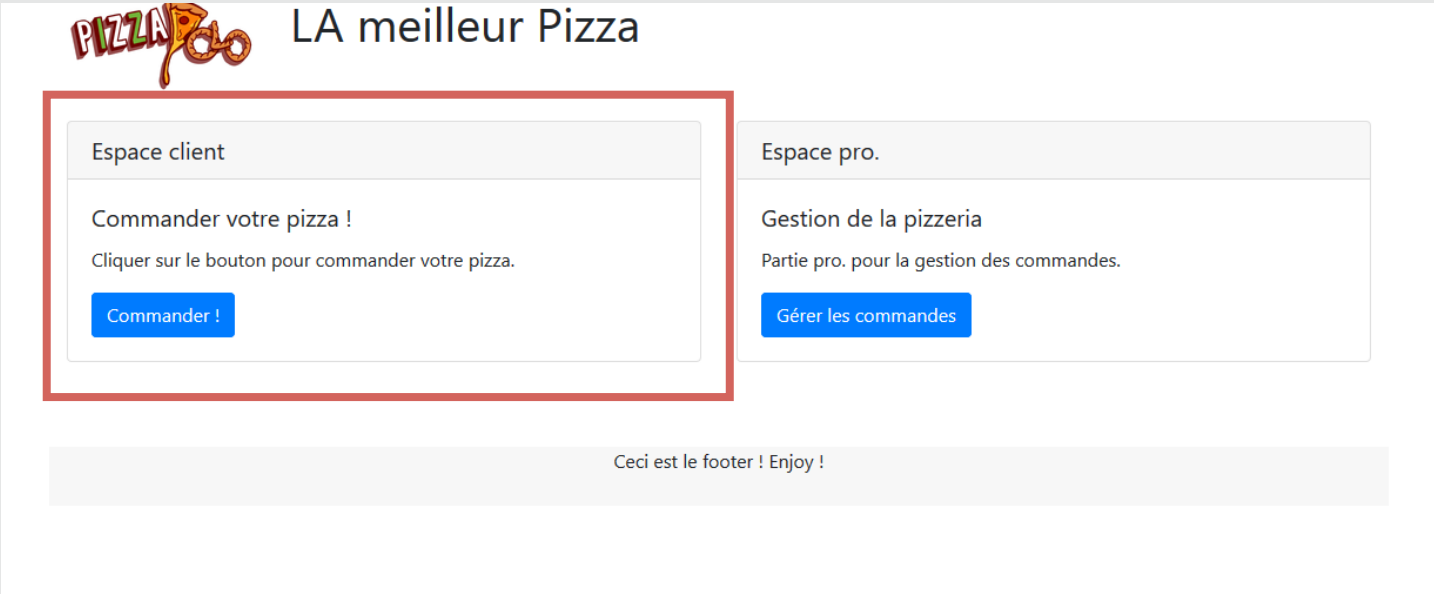
TP

TP01_02-gitConflits

enonce-etape1.pdf

Travaillons ensemble !

- Vous êtes chargé de développer le module « espace client »
- Je suis chargé de développer le module « espace pro. »



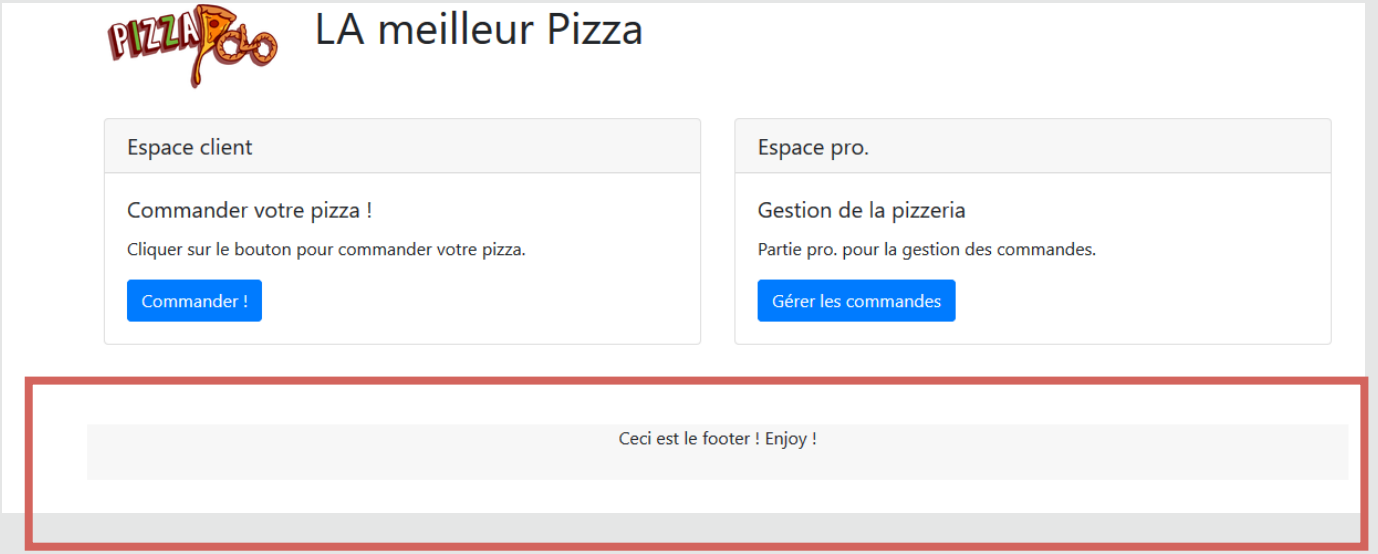
TP

TP01_02-gitConflits

enonce-etape2.pdf

Travaillons ensemble !

- Le chef de projet vous demande de réaliser le footer du site



Peu être est-il un peu fatigué ce chef de projet?...

M02 : GIT-FLOW

La bonne pratique : GIT-FLOW

Git-flow est une convention de travail afin de faciliter et organiser les repo. GIT.

Concepts :

- Un repo est créé avec 2 branches :
  - master → Code source stable
  - develop → Code source en développement

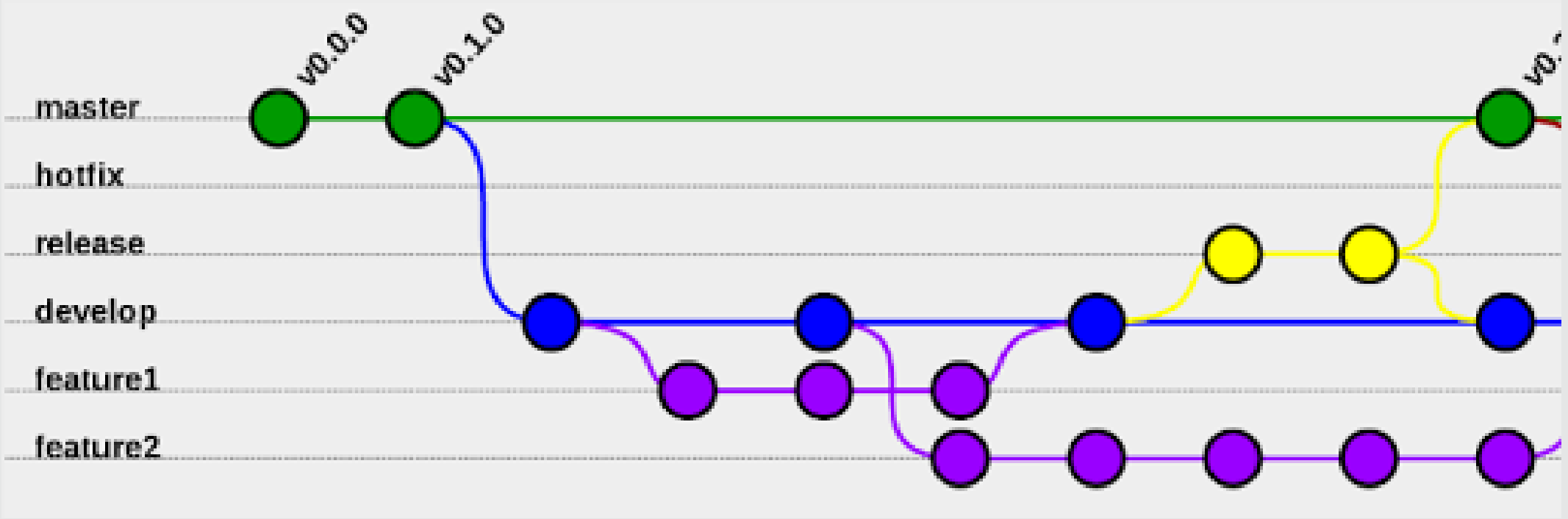
Règles :

- Il est INTERDIT de réaliser des commit sur la branche « master » et « develop »
- Création d'une nouvelle branche à chaque nouvelle fonctionnalité.
  - Préfix de la branche : « feature »
- Création d'une nouvelle branche à chaque nouvelle version.
  - Préfix de la branche : « release » + création d'un Tag
- Création d'une nouvelle branche à chaque correctif de bug.
  - Préfix de la branche : « hotfix »

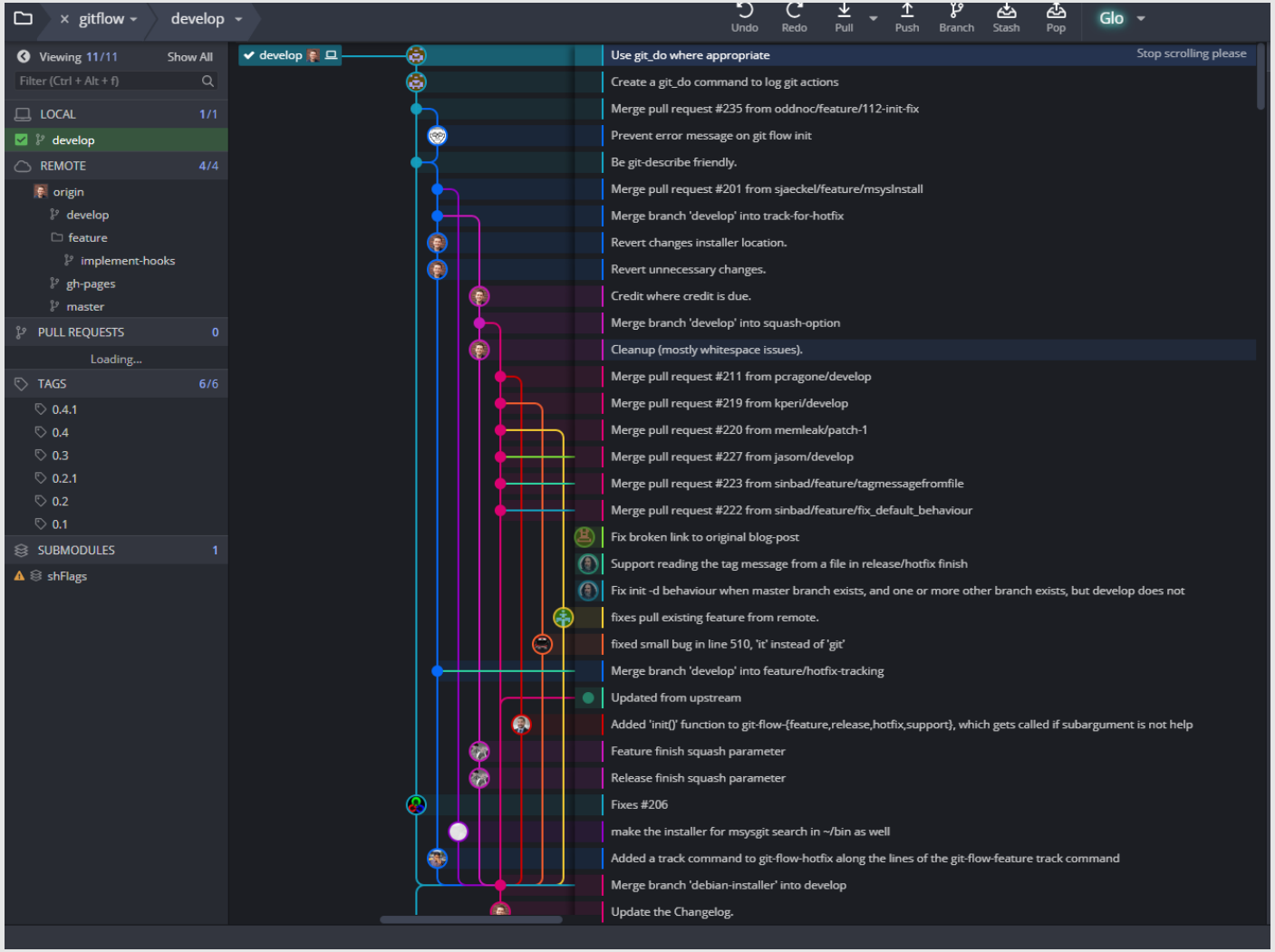
Fonctionnement :

- Le développeur créer une/des nouvelle(s) FEATURE et développe
- Le développeur (ou l'architecte) MERGE la/les FEATURE(S) sur la branche DEVELOP (correction des conflits…)
- L'architecte décide de réaliser une nouvelle version du logiciel :
  - MERGE de la branche DEVELOP sur MASTER
  - Réaliser d'un TAG de la branche MASTER

La bonne pratique : GIT-FLOW



Exemple : https://github.com/nvie/gitflow.git



M02 : GITLAB-CI - PIPELINE

Automatisation de tâches

/!\ GitLab-CI n’est pas GIT !!!

Automatisation de tâches

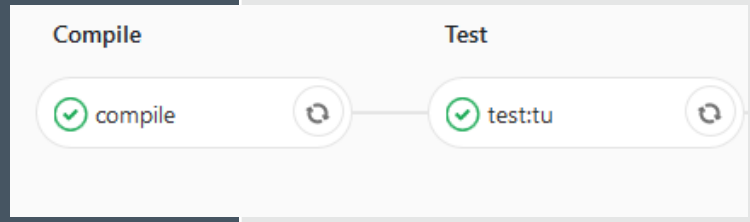
- Nécessite des connaissances en système UNIX et déploiement
- Fonctionne grâce à docker
- Configuration As-Code
- Une tâche est un « stage »

```
image: java:8-jdk

stages:
  - compile
  - test

compile:
  stage: compile
  script:
    - ./gradlew build

test:tu:
  stage: test
  script:
    - ./gradlew test --tests=*
  only:
    - master
```



FIN DU MODULE