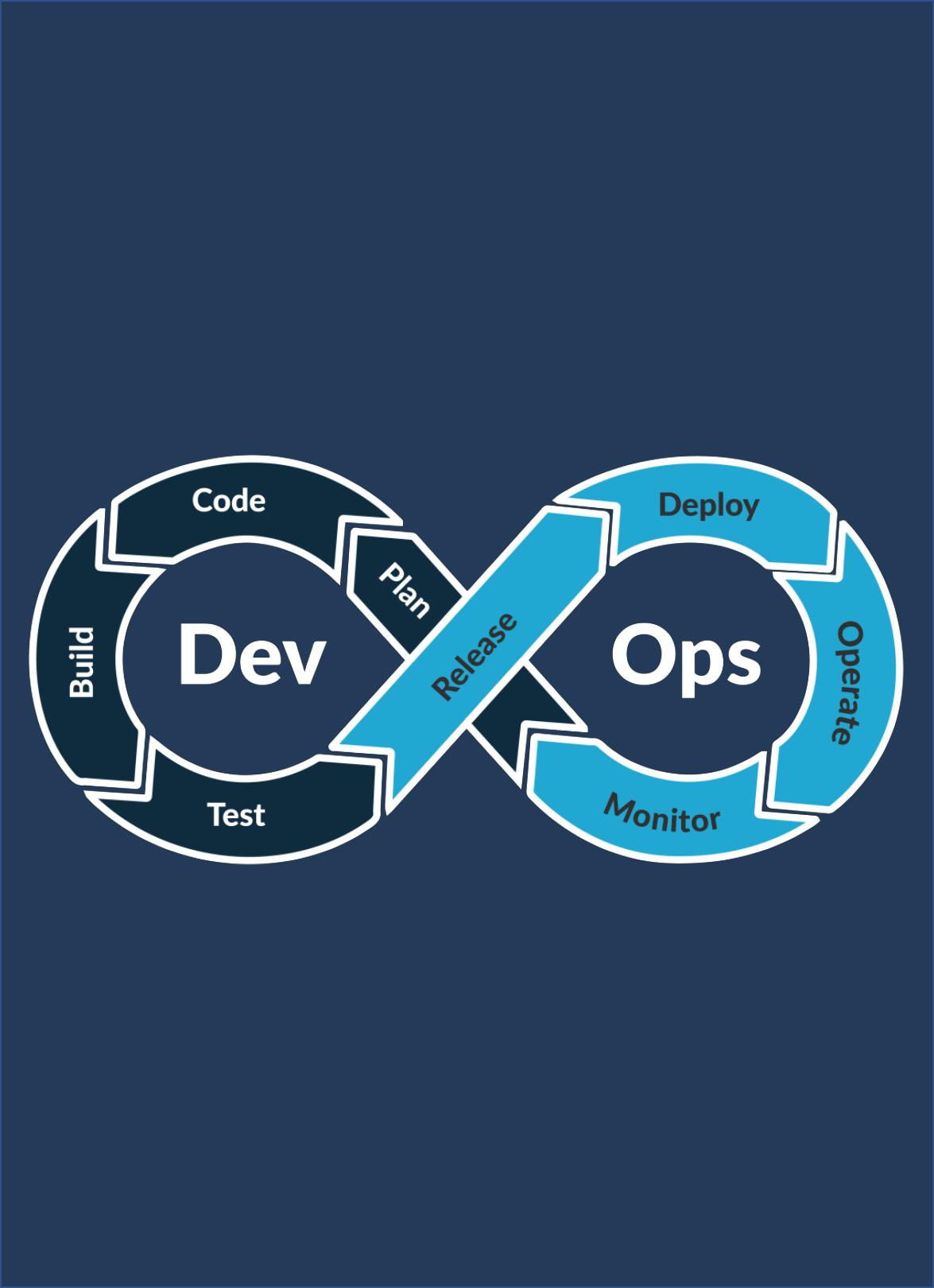
**GIT**

**Aperçu**



**DOFD**

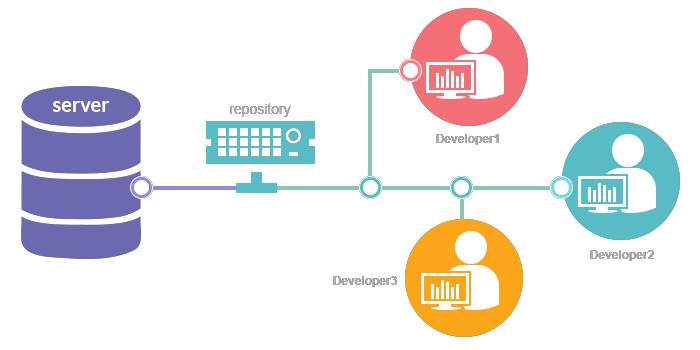
**Programme du module**

**Ordre du jour : Module**

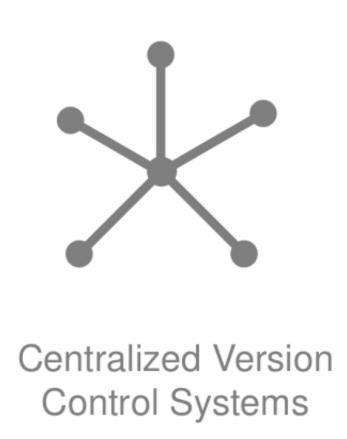
* **Présentation de GIT**
* **Système de contrôle de version distribué**
* **Avantages de GIT**
* **Flux de travail DVCS**
* **POURQUOI Git ?**
* **Fichier GIT Ignorer**
* **Terminologie GIT**
* **Exemples de terminologies GIT**

**Présentation de GIT**

* **Système de contrôle de version(VCS) est un logiciel qui aide les développeurs de logiciels à travailler ensemble et à conserver un historique complet de leur travail.**
* **Vous trouverez ci-dessous les fonctions d'un VCS :**
  + **Permet aux développeurs de travailler simultanément.**
  + **Ne permet pas d'écraser les modifications des autres.**
  + **Conserve un historique de chaque version.**
* **Voici les types de VCS :**
  + **Système de contrôle de version centralisé (CVCS).**
  + **Système de contrôle de version distribué/décentralisé (DVCS).**



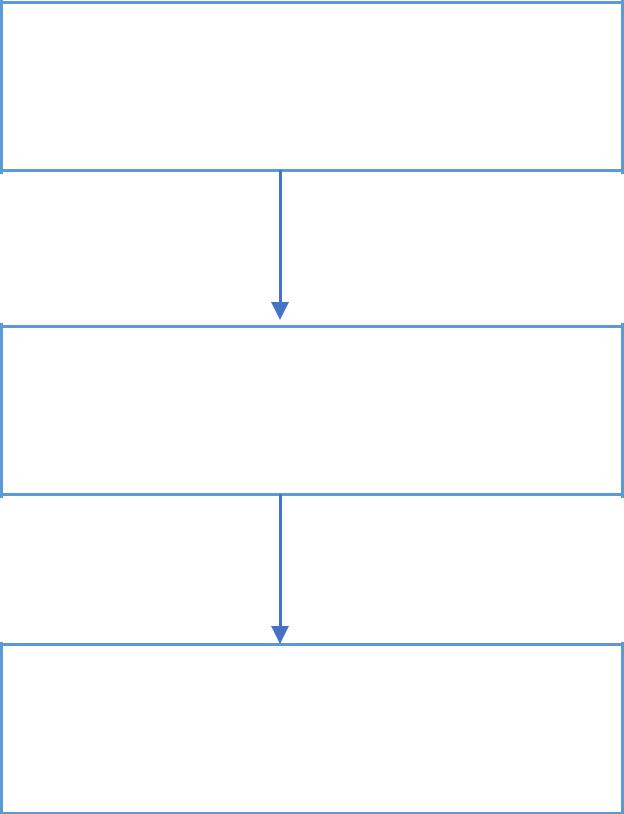
**Système de contrôle de version distribué**



* **Système de contrôle de version centralisé(CVCS) utilise un serveur central pour stocker tous les fichiers et permet la collaboration en équipe. Mais le principal inconvénient de CVCS est son point de défaillance unique, c'est-à-dire la défaillance du serveur central.**
* **DVCSLes clients non seulement récupèrent le dernier instantané du répertoire, mais ils reproduisent également entièrement le référentiel. Si le serveur tombe en panne, le référentiel de n'importe quel client peut être copié sur le serveur pour le restaurer. Chaque extraction est une sauvegarde complète du référentiel.**



**Flux de travail DVCS**



**Répertoire de travail**

**Opération Git add**

**Zone de préparation**

**Opération de commande Git**

**Dépôt Git**

1. **Dépôt local**
2. **Répertoire de travail**
3. **Zone de préparation ou index**

**Étape 1 :Vous modifiez un fichier du répertoire de travail.**

**Étape 2 :Vous ajoutez ces fichiers à la zone de préparation. Étape**

**3 :Vous effectuez une opération de validation qui déplace les fichiers de la zone de préparation. Après l'opération push, les modifications sont stockées de manière permanente dans le dépôt Git.**

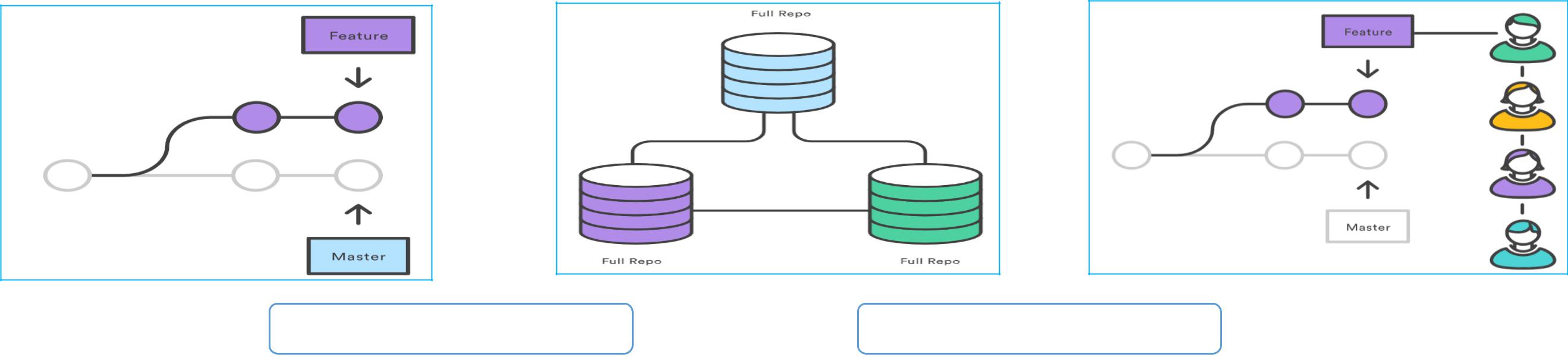
**POURQUOI GIT ?**



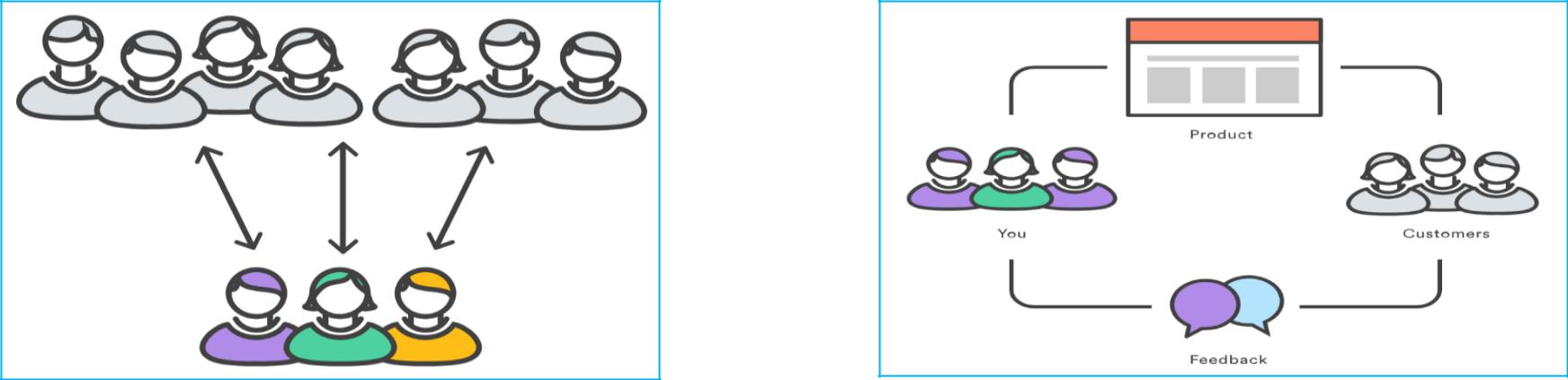
**Distribué** **Demande de tirage**



**Branches de fonctionnalités** **Développement**



**Communauté** **Cycles de publication**



**Avantages de GIT**



**Gratuit et open source (licence libre)**

**et peut être utilisé n'importe où)**



**Sauvegarde implicite(plusieurs copies sont**

**là pour la sauvegarde)**



**Ramification facile**

**Rapide et petit (local rapide et distant)**



**pas besoin de synchroniser souvent, la taille du magasin est**

**petit)**



**Sécurité (algorithme SHA 1)**



**Pas besoin de matériel puissant**

**FICHIER GIT IGNORE**

**-**

**-**

**Un fichier gitignore spécifie les fichiers intentionnellement non suivis que Git doit ignorer. Chaque ligne d'ungitignorele fichier spécifie un modèle**

**Les fichiers ignorés sont généralement des artefacts de construction et des fichiers générés par la machine qui**

**peut être dérivé de la source de votre référentiel ou ne devrait pas l'être**

**engagé**

* **Les fichiers ignorés sont suivis dans un fichier spécial nommé.gitignorec'est**

**enregistré à la racine de votre dépôt. Il n'y a pas de git ignore explicite**

**commande : au lieu de la.gitignorele fichier doit être édité et validé par**

**main lorsque vous avez de nouveaux fichiers que vous souhaitez ignorer**

**TERMINOLOGIE GIT**

1. BRANCH

2. CLONE

3. PUSH

4. PULL

5. COMMIT

6. MERGE

7. TAGS

8. CHECKOUT

9. CONFIG

10. FETCH

11. LOG

12. STATUS

13. ADD

14. STASH

15. FORK

16. SHOW

17. RESET

18. REVERT

19. WATCH

20. STAR

21. MARKDOWN

22. CLEAN

23. BLAME

24. SHORTLOG

25. CHERRY-PICK

26. DIFF

27. AMEND

28. GIT (RM)

29. INSIGHTS

30. HOOKS

**EXEMPLES DE TERMINOLOGIE GIT**



**RAMIFICATION DU FLUX GIT ()**

* **Une branche est simplement une nouvelle version du code source créée lorsque quelqu'un modifie le code source. Branch vous permet de travailler de manière indépendante et vous offre un environnement isolé dans lequel vous pouvez développer sans vous soucier du code source d'origine. Et la branche dans laquelle vous souhaitez fusionner toutes les modifications est appelée branche principale(master) ou branche principale(main), selon le système de contrôle de version que vous utilisez.**
* **Il existe de nombreux types de branches, mais nous pouvons les classer en 2 grandes**

**catégories :**

* **Les branches régulières sont celles que nous utilisons pour les activités quotidiennes et qui sont toujours disponibles dans le référentiel. Les branches sont :Maître, développeur, test QA**

**BRANCHE RÉGULIÈRE**

* **Maître (*maître*)- Il s'agit de la branche par défaut qui est créée une fois qu'un référentiel est créé n'importe où. Tous les codes sont stockés dans cette branche.**
* **Développement (*développeur*)- Cette branche est le premier clone/copie que nous faisons de la branche master. La branche dev permet aux développeurs de créer puis de tester n'importe quel code et, ce faisant, de ne pas perturber notre branche master. Après avoir développé et testé le nouveau code, les développeurs fusionnent la branche dev avec la branche master.**
* **Assurance qualité (*Assurance qualité*)/test branch- Cette branche contient tous les codes de toutes les modifications apportées au**

**code source pour les tests d'assurance qualité. Une fois le test d'assurance qualité terminé, le code est envoyé en production en créant une base de code stable.**

**SUCCURSALE TEMPORAIRE**

* **Branches git temporairessont créées et supprimées selon les besoins. Il s'agit de branches à la demande qui sont créées pour une courte période. Les noms des branches sont Feature, Release et Hotfix.**
* **La branche de fonctionnalité est la copie identique de la branche principale que les développeurs utilisent pour tester**

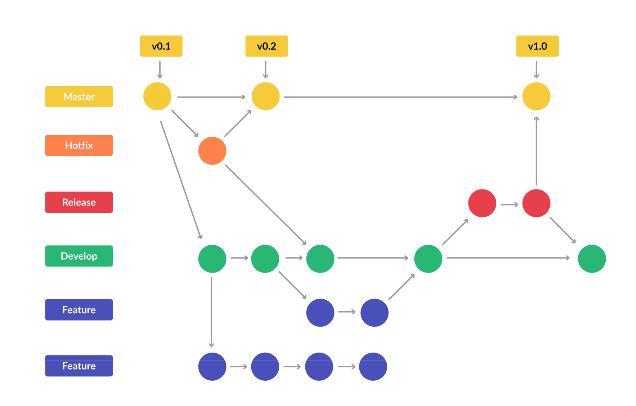
**une fonctionnalité. Les développeurs font une copie de la branche principale, développent de nouvelles fonctionnalités dessus, valident les modifications, ouvrent une demande d'extraction, puis le code est fusionné dans la branche principale après approbation. La méthode de la branche de fonctionnalité est très efficace pour l'environnement d'intégration continue car la branche principale n'est jamais perturbée.**

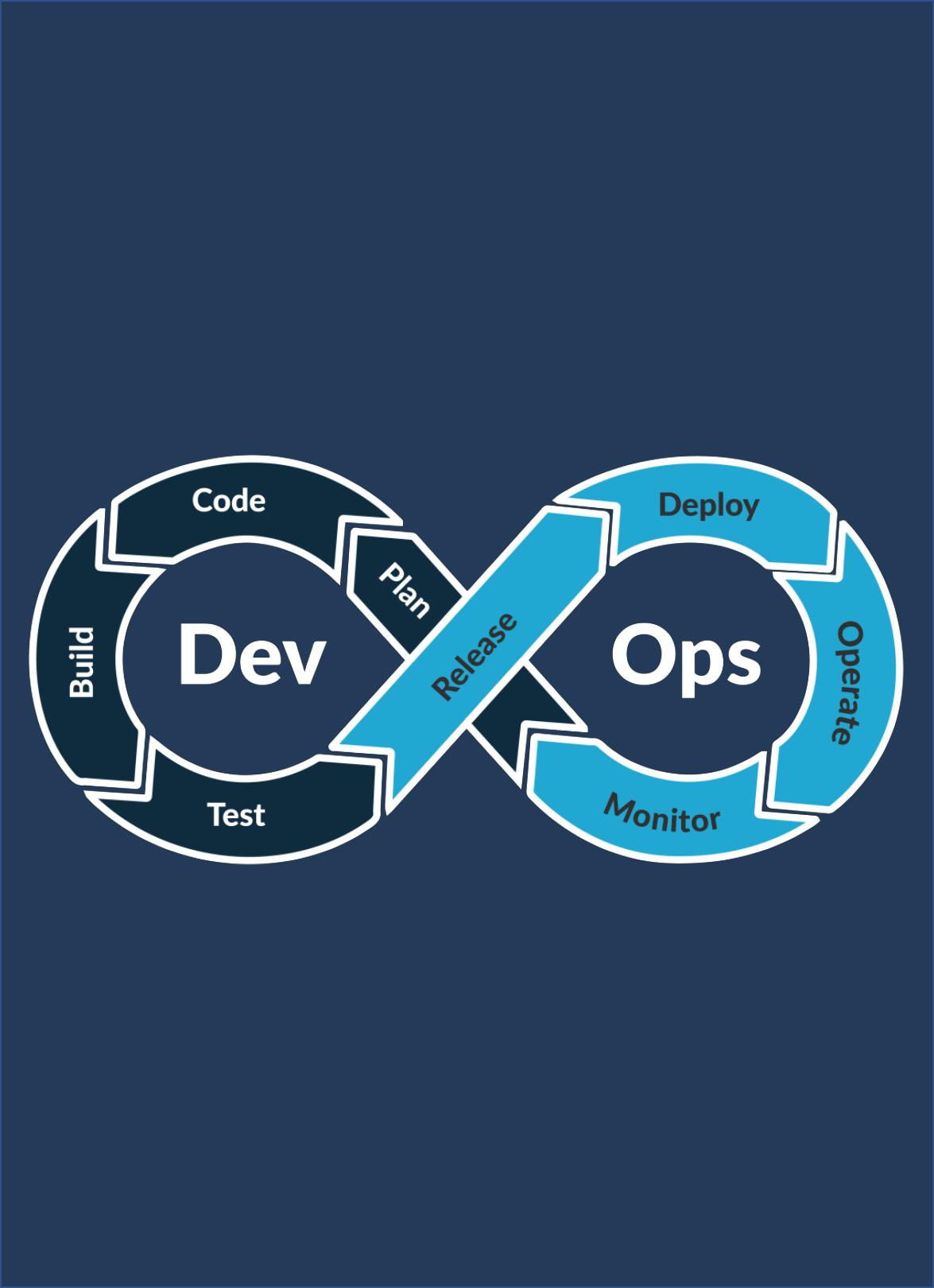
**RAMIFICATION DU FLUX GIT (2)**

* **La création de branches Git est identique à la création de branches de fonctionnalités. Ici, nous faisons une copie de la branche principale, appelons-la Developer Branch, puis nous faisons une copie de la nouvelle branche copiée (Developer Branch) pour créer deux branches distinctes : Release et Feature. Après avoir effectué les modifications et les avoir validées, nous fusionnons la demande dans une nouvelle branche appelée Hotfix Branch.**
* **La branche Hotfix est créée en combinant le code des branches Release et Feature, ce qui permet**

**d'obtenir un code à publier avec la fonctionnalité parfaite. Le but de la branche Hotfix est de corriger tout problème de version ou de fonctionnalité avant de le fusionner avec la branche Developer, puis avec la branche Main.**

**DIAGRAMME DE FLUX GIT**





**Suivant:**

**Jenkins**