Git

Sommaire

Git, à quoi ça sert ?

Git est un **outil de versionnage**. Il permet :

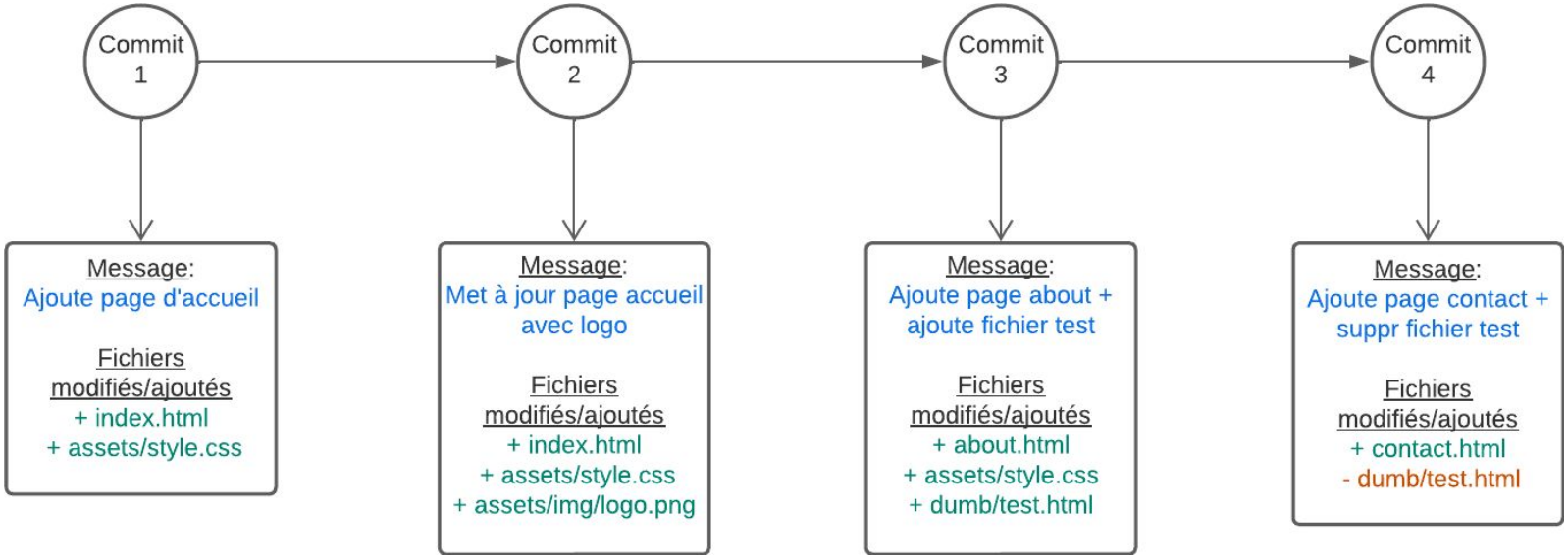
- De surveiller et garder en mémoire les changements effectués sur des fichiers
- Permet de retourner en arrière sur les versions d’une application
- Permet de coordonner le travail entre plusieurs développeurs
- Permet de consulter les personnes qui ont modifiées un fichier et quand
- Peut s’utiliser avec des répertoires à distance pour faciliter l’échange de travail

Git, techniquement comment ça fonctionne ?

Avec git, on réalise des **commits.** Les commits contiennent les informations de changements du contenu d'un ou plusieurs fichiers.

- On choisit quand est-ce que l'on fait un **commit**
- On choisit les modifications d'un fichier que l'on souhaite mettre dans le **commit**
- Un historique se construit et nous permet de se déplacer de **commit** en **commit**
- Possibilité d'héberger ce dossier/cet historique en ligne sur Github ou Gitlab

Schématisation commits



Installation

Télécharger l’installer sur le lien suivant :

https://git-scm.com/downloads

Le lancer et suivre les instructions

Git, comment ça fonctionne ?

Git est un outil en ligne de commande, il s'utilise donc dans un terminal via des commandes. Voici les commandes de base :

git init	Initialiser Git dans le dossier courant, génère un dossier **.git**
git add <fichier>...	Indiquer le/les fichier(s) spécifié(s) à mettre dans le prochain commit.
git add .	Indiquer que tous les fichiers modifiés ou créés sont à mettre dans le prochain commit.
git rm --cached <fichier>...	Retirer le/les fichier(s) spécifié(s) du prochain commit.
git rm --cached -r .	Retirer tous les fichiers du prochain commit.
git status	Vérifie le statut des fichiers du répertoire courant
git commit -m “Titre du commit”	Créée un commit avec tous les fichiers en “stage”
git push	Envoyer tous les commits sur un répertoire Git en ligne
git pull	Récupérer tous les nouveaux commits d'un répertoire Git en ligne
git clone	Cloner un répertoire en ligne dans le dossier local courant

Initialiser Git sur un projet existant

On imagine avoir un projet en local. On souhaite y ajouter la gestion des versions via Git.

Première chose à faire dans un terminal à l'emplacement de votre dossier de projet :

```
git init
```

Suite à cette commande le dossier sera “surveillé” par Git et permettra de garder en mémoire plusieurs versions d’un fichier. On pourra ensuite enchaîner les commandes suivante (enchaînement fréquent quand l’on souhaite commit) :

```
git add .
git commit -m “Commit d’initialisation”
```

Si vous disposez d’un repertoire distant (Sur Github ou Gitlab) :

```
git remote add origin urlDeMonRepoDistant.git
git push
```

Récupérer un projet existant depuis Github ou Gitlab

Si l’on souhaite récupérer un projet où le code est hébergé sur Gitlab ou Github il faut utiliser la commande suivante à l’endroit où l’on souhaite télécharger le dossier du projet :

```
git clone urlDeMonRepoDistant.git
```

Récupération des dernières mises à jour du répertoire distant

Si l'on travail avec un répertoire distant (sur Github ou Gitlab) on peut récupérer les nouveaux commits via la commande :

```
git pull
```

Par contre, si un fichier à été modifié à la fois sur les commits du répertoire distant et chez vous en local, il va falloir fusionner les deux versions, c'est ce que l'on appelle un **merge conflict**, ce sera à vous de choisir comment fusionner.

*Petit conseil : faites toujours un **git pull** avant de commit + push sur une branche distante, au cas où un de vos collègues aurait envoyé des commits entre temps.*

Annulation des modifications locales

Si vous avez apporté des modifications sur un fichier et que vous ne souhaitez plus les garder et revenir à l'état initial par rapport au commit précédent, faites :

```
git checkout cheminVersMonFichier
```

Si vous souhaitez revenir en arrière à un commit précis et supprimer toutes les modifications faites après ce commit, utilisez :

```
git reset –hard numeroCommit
```

Utilisez cette dernière commande avec précaution !

Commandes utiles !

Si vous souhaitez voir les fichiers actuellement indiqués comme “staged” et qui seront donc ajoutés au prochain faites la commande suivante (à faire avant un **git add** ou **git commit** par exemple) :

```
git status
```

Si vous souhaitez voir l’historique des commits l’une des deux commandes :

```
git log
```

```
git log –graph –oneline
```

Si vous voulez connaître les modification apportées par un commit :

```
git show ID_DU_COMMIT
```

Si vous souhaitez voir les modifications réalisées sur un fichier comparé au dernier commit :

```
git diff chemin_vers_mon_fichier
```

Les branches

Les branches permettent de travailler sur des versions de code qui divergent de la branche principale.

La branche principale, par convention, s'appelle **main.**

Pour lister les branches existantes :

```
git branch
```

Pour créer un branche, nommée **develop** par exemple :

```
git branch develop
```

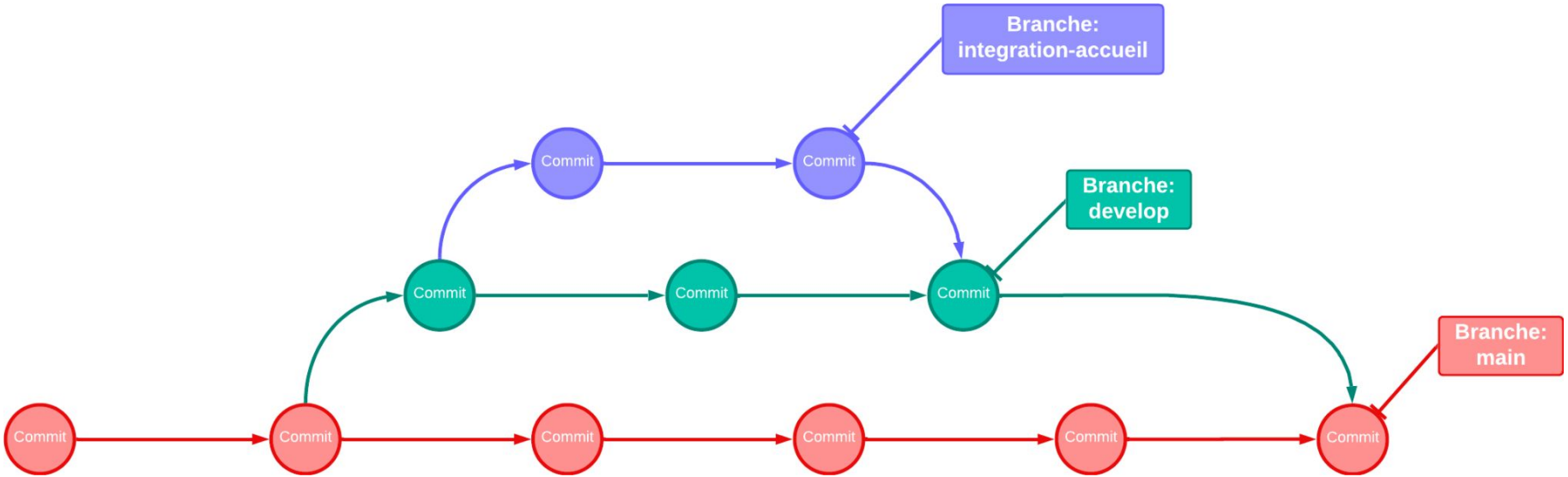
Pour se placer dans cette branche et créer de nouveaux commit dessus :

```
git checkout develop
```

Pour supprimer un branche :

```
git branch -d develop
```

Schéma branches



Pourquoi utiliser les branches ?

Les branches servent à organiser le développement, en général on essaie de se créer une branche pour chaque fonctionnalité que l'on souhaite développer, cela nous permet de ne pas entrer en conflit avec d'autres développeur qui développent eux aussi une fonctionnalité.

Une fois qu'une branche n'est plus utile car le développement de la fonctionnalité est terminé, soit on passe par une **merge request** pour une revue de code soit on merge directement la branche :

```
git merge develop
```

Cette commande merge la branche **develop** dans la branche courante (ex: **main**)

Le fichier .gitignore

Souvent vous ne voulez pas que certains fichiers/dossiers soient envoyés soit versionnés et envoyés sur les répertoires distants, dans ce cas, on peut créer un fichier .gitignore à la racine du projet ou git est initialisé.

On y écrit à l'intérieur les fichiers que l'on ne souhaite pas voir versionnés :

```
# .gitignore

*.txt
db/*.sql
var/cache/
.env.local
node_modules/
.idea/
vendor/
```