[SUPPORT] Git & GitHub introduction

Présentation

Tous développeurs doit mettre en place un système lui permettant d'optimiser son travail : après avoir passer plusieurs jours à développer un module spécifique pour un projet il faut être capable de le réutiliser en quelques cliques pour un autre projet. Depuis toujours, nous avons mit en place des solutions plus au moins solides mais il est aujourd'hui possible d'utiliser un système globale et largement répandu : Git et GitHub.

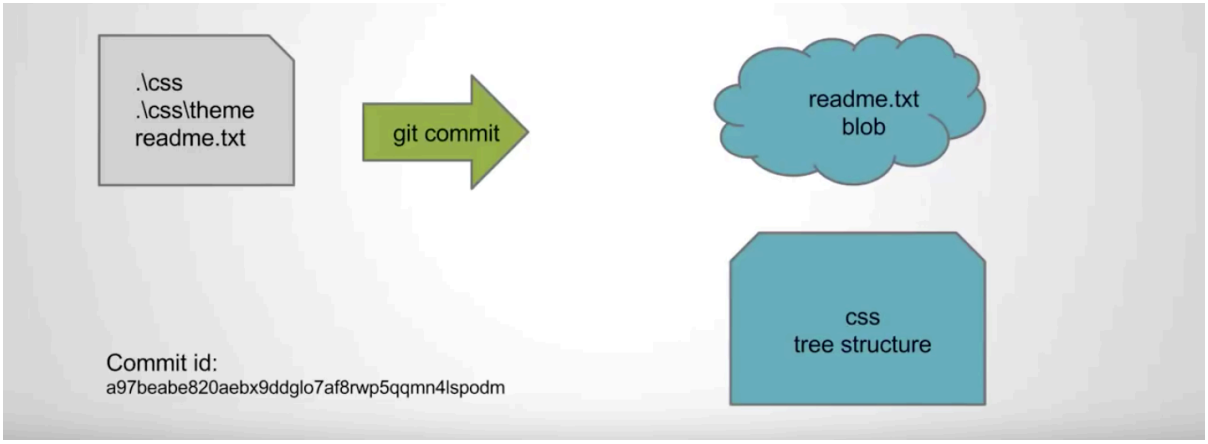
Git est une extension à installer sur votre machine depuis le site officiel qui permet de gérer les versions en locale. GitHub est une plateforme en ligne dans laquelle nous pouvons sauvegarder nos versions pour garder une trace de noter travail en ligne mais également pour faciliter le travail en groupe dans la mesure ou l'on peu gérer les collaborateurs du projet.

Références

Site officiel Git : https://git-scm.com/
Vidéo Tuto : https://www.youtube.com/watch?v=rP3T0Ee6pLU&list=PLjwdMgw5TTLXuY5i7RW0QqGdW0NZntqiP
Vidéo Tuto : https://www.youtube.com/watch?v=5IcYILdejs8

Core concept

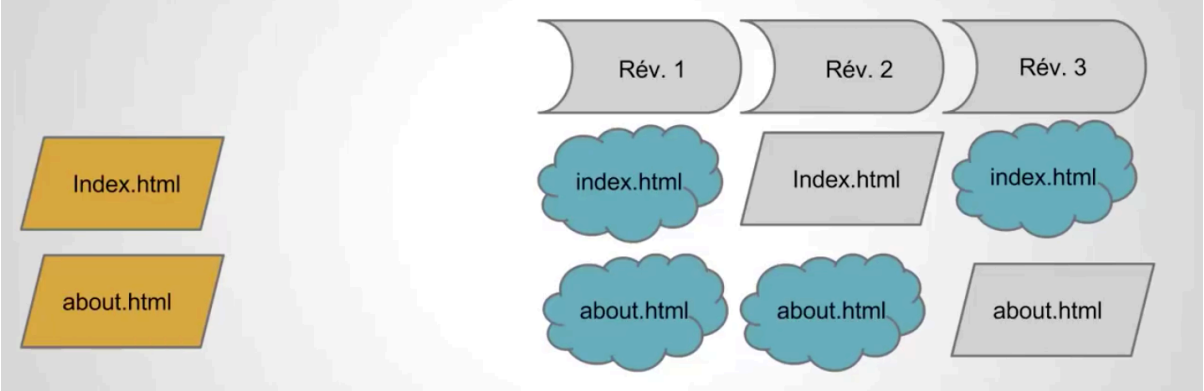
Les Commits (sauvegardes)



« Faire un commit » est l'action qui permet de sauvegarder en local la version actuelle des fichier « commité ». Une fois sauvegarder, nous pouvons modifier les fichiers avant de faire un nouveau commit : nous aurons alors deux versions différentes des mêmes fichiers ce qui nous permet de conserver l'historique des modifications

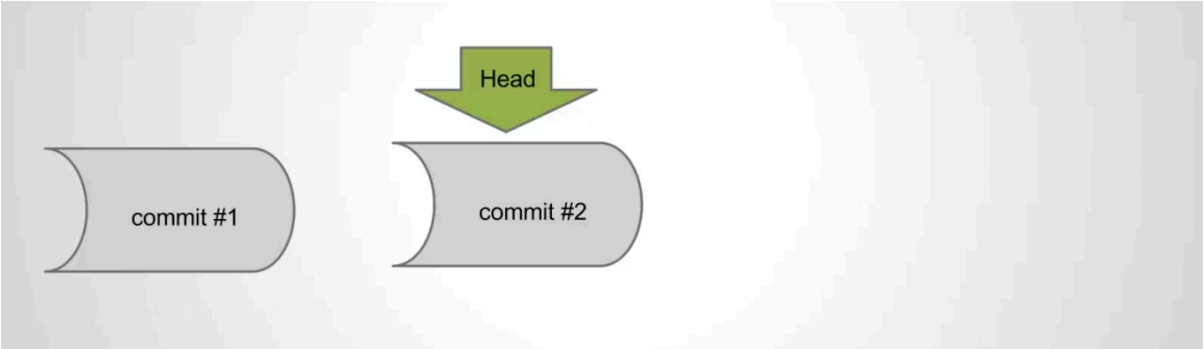
---

Les Pushs (révisions)



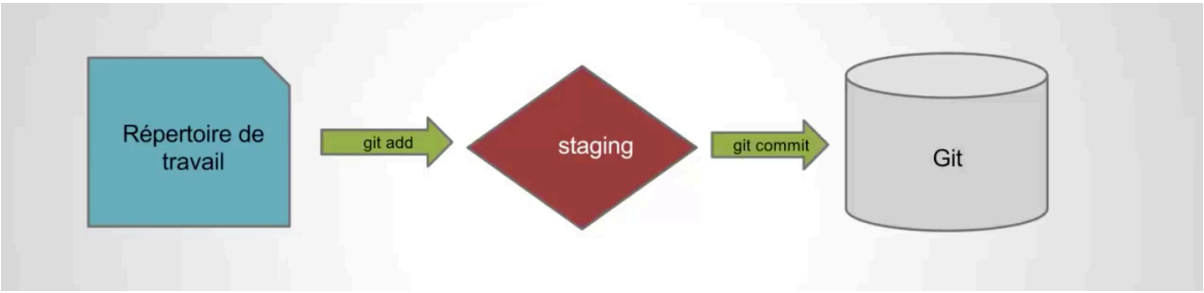
Une fois nos commits réalisés, nous allons arrivé au moment ou le travail en cours est validé, il faut donc sauvegarder tout notre dossier pour l'envoyer sur GitHub. Cette action s'appelle « un push » car nous envoyons la totalité de notre dossier ainsi que l'historique des changement dans un dossier (repository) sur GitHub.

Le HEAD (position temporale)



Lors des push ou des commit, il est possible de rencontrer un conflict : git ayant vérifier tout les fichiers, il peut vous indiquer que tel ou tel fichier contient des changement qui ne sont identiques entre les version. Il est alors possible de vérifier le head (position temporelle) dans les fichiers concerner pour sélectionner la version de code à conserver.

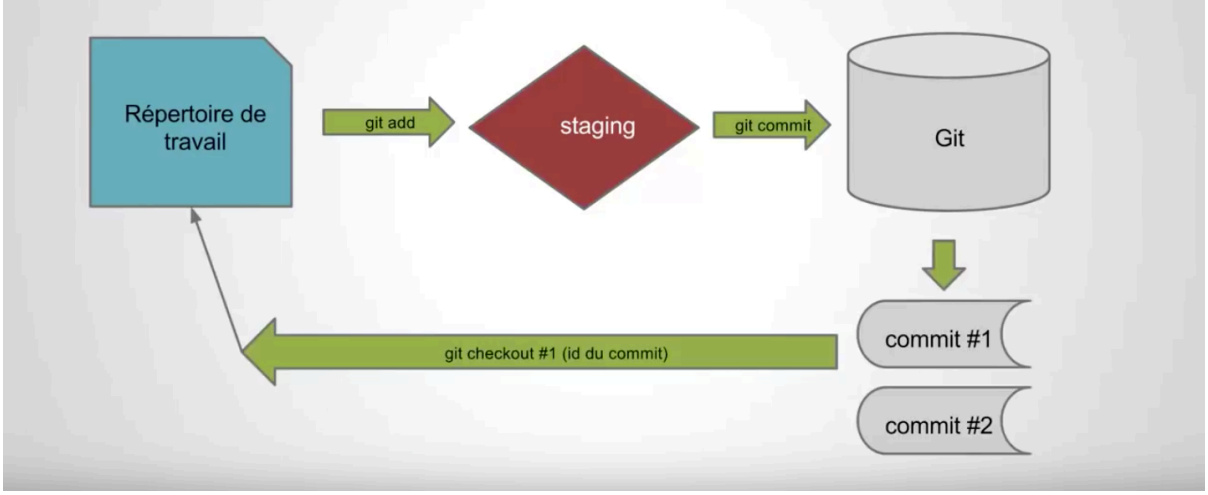
Le staging (snapshot de la version)



Le staging est l'action qui consiste à enregistre en local une version de nos fichier. Ce staging permet lors du commit de vérifier la version en cours avant de la « commiter ». Chaque fichier doit être dans le stage pour pouvoir les sauvegarder

---

Le checkout (gestion des versions)



La commande checkout permet de voyager dans le temps sur notre dossier. Nous pouvons sélectionner un des commits du projet pour faire en sorte de revenir à la version de commit.

Les commandes Git

Il est possible d'utiliser Git et GitHub de deux façons différentes : soit en utilisant les logiciels GitHub, soit en passer en ligne de commande. La ligne de commande est la solution la plus efficace car elle vous permet de gérer Git très précisément alors que les logiciels ne propose qu'une partie des solutions Git.

Commandes générales Git

- Liste des commandes git : git help
- Initier un projet git : git init
- Voir la configuration : git config —list
- Afficher la liste de commits (sur une ligne) : git log —oneline
- Afficher les commits d'un fichier spécifique : git log -p index.html
- Afficher les modifications apportées aux fichiers : git diff
- Regarder l'état d'un commit : git checkout 1234567
- Modifier un fichier dans un commit : git checkout 1234567 index.html
- Défaire un commit : git revert 1234567

- Défaire les modifications du commit d'un fichier spécifique : git revert 1234567 index.html
- Revenir à un commit précis : git revert 123456 (options —soft, —hard, —mixed)

Configuration d'un dossier

- Initialiser git : git init
- Configurer l'email : git config --global (ou --local) user.email «email@user.com»
- Configurer le nom : git config --global (ou --local) user.name «nom»
- Configurer les couleurs des commande : git config --global (ou --local) color.ui true
- Vérifier la config : git config —list
- Voir l'état du dossier : git status

Configuration un nouveau projet

- Créer la fichier readmet.md : touch readme.md
- Créer un fichier index.html : touch index.html
- Ajouter le ficher au « stage » : git add nomduficher.ext
    - Ajouter tous les fichiers par type : git add *.html
    - Ajouter tous les fichiers : git add —all
- Envoyer les modifications : git commit -m «Mon premier commit»
- Envoyer tous les fichiers modifier : git commit -a -m «Commit de tous les fichiers»
- Créer un fichier pour supprimer ignorer des fichier : touch .gitignore
    - Ouvrir le fichier et ajouter ligne par ligne les fichier à ignorer (ou les type de fichier *.tmp)

Gestion des branches

- Créer une branche : git checkout -b nomDeLaBranche
- Voir la liste des branches : git branch -a
- Travailler sur la branche : git checkout nomDeLaBranche
- Envoyer les commits d'une branche dans le master : git rebase master
- Ajouter les modification d'une branche dans le master : git merge nomDeLaBranche

Gestion des remises

- Sauvegarder des modifications sans commit : git stash
- Voir les éléments sauvegarder en mémoire : git stash list
- Ajouter la version sauvegardé au commit en cours : git stash apply
- Supprimer la version sauvegardée : git stash drop

Gestion des dépôts distants

- Configuration d’un dossier local : git int —bare
- Définir le remote du projet : git remote add nomDuRemote (convention 1er : origin) adresseDURemote
- Renommer un remote : git remote rename ancienNom nouveauNom
- Supprimer un remote : git remote remove nomDuRemote
- Voir la liste des dossiers remote: git remote -v
- Voir la liste des branches disponibles dans le remote : git branch -r
- Envoyer une branche dans le remote : git push nomDuRemote nomDeLaBranche
- Récupérer une branche depuis un remote : git pull nomDuRemote nomDeLaBranche