# Bases de GIT

## Intro

Hey bien salut à tous c'est **Graven** pour une nouvelle video sur les `bases de l'outil GIT` .

Dans cette vidéo vous allez apprendre: 1. Qu'est-ce que GIT ? 2. Comment cet outil fonctionne 3. Et comment s'en servir facilement.

## Bases

**Bien commençons par le commencement.**

La première question que l'ont en droit de se poser EST

**Qu'est-ce que GIT ?** Et bien il s'agit ni plus ni moins que d'un logiciel, d'un outil permettant de controler la ou les versions d'un projet informatique.

## Exemple

Prennons un exemple très simple pour essayer de bien comprendre Hugo 16 ans, est un jeune licéen devant réaliser un site web pour son école

**Une fois le top lancé par son professeur** il commence à créer un premier fichier contenant les premieres balises de sa page et viens enregistrer son travail dans un fichier index.html qu'il ' `enregistre sur son ordinateur` . La journée passe très vite Hugo est fatigué est va se coucher.

**Le lendemain** Hugo se reveille réouvre son ordinateur et décide de continuer sa page en ajoutant 1 bouton ainsi qu'un nouveau fichier pour commencer à styliser sa page.

Il créer donc un fichier styles.css qu'il `enregistre à nouveau sur son ordinateur` .

Le voila avec un projet qui commence à bien avancer ! Hugo est tres content. La journée passe et la nuit tombe vite en cette période de confinement.

**Le 3e jour** Hugo se reveille ouvre son ordinateur Et la il se reçoit un message de son proffesseur lui indiquant qu'a partir de maintenant, il devras travailler à son camarade Mickael qui s'occupera de la partie design.

Hugo se dit génial je vais envoyer le code à Mickael pour qu'il continue son projet et il me renverras le fichier une fois finit.

Les journées passe et Hugo se rend compte qu'il se retrouve bloquer dans son code tant que son camarade n'a pas finit sa partie.

De plus, Mickael revient vers lui et lui dit qu'il a perdu tout son travail et qu'Hugo va devoir lui redonner le fichier.

## Bilan

Bilan, pleins de problemes sont apparus pendant cette periode de developpement comme

- le fait qu'Hugo ait du attendre le code de Mickael pour continuer sa page.
- ou encore que Mickael ait remplacé son code par inadvertance

Tout ces problèmes aurait pu etre résolu d'un claquement de doight si nos confreres aurait utilisé l'outil GIT.

---

# Avec GIT

Remontons un peu le temps en utilisant cette fois ci les bonnes pratique.

Hugo 16 ans, est un jeune licéen devant réaliser un site web pour son école,

**Une fois le top lancé par son professeur**

## 1. Notre élève va pouvoir télécharger la derniere version de GIT.

1. Il se rend dans la `description de la video` de son youtubeur favoris pour acceder au lien vers le site officiel de l'outil GIT
2. Sur cette page il lui suffit de `cliquer sur le bouton pour acceder à la derniere version` pour son systeme d'exploitation
3. Enfin une fois le telechargement fini, il s'agit d'une installation classique, donc il lui `suffit de passer les differents étapes` et git sera désormais installé sur son ordinateur

## 2. Setup du projet

La seconde etape va etre de créer un espace qui contiendras l'ensemble des fichiers du projet.

Cet espace hebergé à distance s'appelle un `repository` . ou `dépot` en français.

Il existe plusieurs site pour en créer un. L'uns des plus connut est GITHUB. Hugo va pouvoir

1. se rendre sur le site et `créer un compte` .
2. puis il lui faudra cliquer sur `new repository` pour créer un `nouvel espace` .

En specifiant : 1. le nom du dépot (par exemple `SiteEcole`) 2. Une description si le souhaite `Ma premiere page web pour le projet scolaire` 3. Si il veut que le projet soit `Publique` ou `Privée` 4. Il peut également cocher l'option `README` qui est une `petite documentation` du projet qui va generer un fichier pour ajouter des instructions supplementaires.

5. Enfin il lui suffit de cliquer sur `Create repository` et le `depot` apparait !

Parfait ! La configuration de son espace est presque finit ! Seulement, notre étudiant n'ayant pas encore accès à cet espace. Il lui faudras créer une `copie` sur son ordinateur (en local)

#### 2.5 Cloner le projet

Pour cela : 1. il va devoir se rendre `sur son bureau` 2. et ouvrir l'`invite de commande de GIT` en faisant un clique droit droit puis GIT Bash. 3. C'est sur cet outil qu'Hugo et Mickael vont pouvoir ecrire `des commandes` pour `envoyer`, `recevoir` ou `ajouter` de nouveaux fichiers à leur espace.

4. Nous allons lui proposer de taper la commande `git clone lien_du_depot`

Cela fera apparaitre un dossier contenant `une copie` des elements du `dépot` en local, c'est à dire sur notre ordinateur.

## 3. Début du projet

Super ! Hugo va pouvoir se mettre au travail

#### Jour 1.

Il dédide de commencer par la création d'un premier fichier contenant les premieres bases de sa page et viens enregistrer son travail dans un fichier index.html qu'il `enregistre dans cette version local du dépot`.

Seulement, avant de quitter son poste. Hugo va devoir `envoyer les changements` sur github car pour l'instant le `code est encore sur son ordinateur`.

Il va donc devoir revenir sur le terminal Et taper la commande `cd SiteEcole` qui va lui permettre d'aller dans le dépot local du projet

Une fois à l'interieur il sera possible d'ajouter son nouveau fichier "index.html" en faisant la commande

`git add index.html` si a plusieurs fichiers il peut aussi tout les ajouter en faisant `git add *`

Cela va simplement `ajouter ce fichier à une liste en attente d'envoi`.

Une fois sa liste de fichiers prete à l'emploi Hugo va pouvoir `empaqueter` toutes ces `modifications` dans ce que l'on appellera un `commit`,

C'est un peu comme une boite qui va contenir tout les changements correspondant à une version, (ajout d'un nouveau code, suppresion, modification d'une ligne, etc, etc)

Il ne nous resteras plus qu'a executer la commande `git push` Qui permettra l'envoi de ce `commit` sur son dépot en ligne cette fois ci.

Parfait ! Hugo peut désormais se deconnecter en toute tranquilité son projet a été sauvegarder !

#### Jour 2.

**Le lendemain** Hugo se reveille réouvre son ordinateur et décide de continuer son projet en ajoutant un nouveau bouton ainsi qu'un nouveau fichier pour commencer à styliser sa page.

Il créer donc son fichier index.html et en créer un nouveau nommé styles.css qu'il `enregistre à nouveau sur ce dépot local`.

Avant de quitter son poste il reproduit les memes étapes qu'hier

1. En commençant par un coup de `git add *` pour mettres les nouveaux fichiers
2. Puis genere un nouveau commit nommé ayant pour description `Premiers design` grace à la commande `git commit -m Premiers design`
3. Il ne lui reste plus qu'à `publier` les changements sur le dépot en faisant à nouveau la commande `git push` et cela s'envoi à nouveau parfaitement et genere une nouvelle version actualisé du projet.

La fatigue se fait sentir, notre etudiant va se coucher

## 4. Projet en groupe

**Le 3e jour** Hugo se reveille ouvre son ordinateur Et la il se reçoit un message de son professeur lui indiquant qu'a partir de maintenant, il devrait travailler avec son camarade Mickael qui s'occuperais de la partie design.

Hugo se dit pas de problème, il n'a qu'à faire installer GIT à son ami Mickael.

Lui faire copier le projet avec la commande `git clone` Et les voila tout les deux pret pour travailler ensemble !

Effectivement à partir de maintenant les deux amis peuvent chaquer travailler dans leurs coins et envoyer leurs commits sur la base en ligne pour faire avancer leurs projets.

Seulement que va t'il se passer si les deux eleves effectue des modifications sur le meme fichier ?

Et bien rien de bien enthousiasme, pleins d'erreurs vont apparaitre Le code risque de ne pas contenir toutes les fonctionnalités, un des deux elves va se retrouver avec son code remplacés.

Bref, c'est compliqué. Les deux camarades vont donc devoir travailler ensemble et communiquer sur le projet en question.

La bonne pratique serait que lorsque Hugo à finit de faire son `git push`, Mickael recupere les nouveautés en faisant la commande `git pull` qui lui permettras

d'absorber ces nouvelles lignes avant qu'ils puissent lui à son tour envoyer son code.

Mais la vous allez me dire ok graven c'est bien beau tout ça, mais meme si le code a bien était récuperé qu'est-ce qui nous garantit que des lignes n'ont pas été supprimé par Hugo ?

Et bien en réalité rien. C'est pour cela que Mickael va devoir comparer son fichier avec celui d'Hugo afin de reajuster les modifications communes, et enlever les "conflits de commits" qui seront apparut. Tout simplement.

En vrai c'est un peu plus compliqué, mais on va rester simple pour cette vidéo. Ok :)

## 5. Intro branche

Parfait on a donc bien résolu le probleme de nos deux etudiants !

Alors on pourrait se dire que cela est suffisant et qu'ils ont désormais tout les outils en main pour avancer sur le super site web.

Seulement ils n'ont pas encore touts les bases pour faire un gestion de version de qualité.

Essayons analyser ce qui se passe lorsque les deux eleves collaborent

Lorsqu'Hugo envoi son `push` il publie une nouvelle version de l'application vous etes daccord avec moi sur ce point.

De meme lorsque `Mickael` procede à la meme commande , cela vient se rajouter comme une nouvelle version du projet, et ce sans pour autant écraser la version précedente.

Cette ligne qui s'incremente au fur et à mesure c'est ce que l'on appelle `une branche` c'est en quelque sorte la ligne de vie de votre `projet`

Par défaut la branche généré à la creation du projet s'appelle la branche `master` c'est la piece maitresse du dépot celle qui comporte le code final du projet.

## 6. Séparation Branches

Donc dans un premier temps il serais plus judicieux de créer plusieurs branches pour bien identifier les étapes de conception du projet.

Tout dabord on a dit que branche `master` sera la branche final du projet Donc par défaut Hugo et Mickael ne devront rien mettre dessus tant que leur projet n'est pas fini.

Ils devront donc s'organiser pour créer une autre branche pour la partie developpement.

Si c'est Hugo qui s'en charge il va devoir effectuer differentes commandes

Commande n°1 : `git branch dev` pour créer une nouvelle branche nommé `dev` Commande n°2 : `git checkout dev` pour se rendre dessus Et enfin `git push --set-upstream origin dev` pour envoyer publier la nouvelle branche sur le depot en ligne

Ensuite pour que Mickael puisse se rende dessus il va devoir absorber les nouvelles branches du projet en faisant `git fetch` Cela lui permettre de permettre à jour le dépot présent sur son ordinateur. Ce qui evite de devoir supprimer tout son dossier pour refaire un clone du projet.

Une fois le projet rafraichit il peut a son tour faire `git checkout dev` pour se placer dessus et continuer le travail en publiant désormais son travail sur cette nouvelle branche. Parfait.

Maintenant qu'une branche de travail existe, on pourrait se dire, très bien il peuvent continuer leur code sans probleme.

Et c'est la qu'on va devoir les arreter à nouveau dans leurs demarches car ils sont encore une fois sur la meme branche et

si Mickael dédide de publier son code en creant un nouveau commit Lorsque Hugo va vouloir repartir de son code il va recuperer la derniere version publié par Mickael ils ne pourront donc pas travailler en simulatnnée dans de bonnes conditions.

## 6. Séparation Branches 2

Pour pallier à cela nous allons encore diviser les branches mais cette fois ci par fonctionnalité.

Si Hugo travaille sur le menu de navigation du site, alors il va créer une nouvelle branche qui va s'appeler `features/navmenu` par exemple.

Pour ce qui est de Mickael travaillant sur la feature de la gallerie photos, il devras créer sa propre branche nommé `features/gallery` par exemple.

Maintenant que le travail est bien séparé chacun peut evoluer sur sa branche Et publier des modifications sur la fonctionnalité sur laquelle il travaille. Parfait.

Enfin, lorsque tout deux auront fini leurs travails. Ils obtiendront chacun un commit final et pourront `fenvoyer ça sur la branche dev` fpour publier une premiere version beta de leur site web

c'est ce que l'ont appelle `fun merge` f, ça permet de fusionner deux branches pour assembler plusieurs fonctionnalités.

Ont peut ainsi publier sa premiere version sur `dev` puis Hugo et Mickael vont à nouveau travailler sur une feature. Ils publient, ils publients, Ils refont une fusion, un `merge`

Et une fois qu'ils identifie qu'une version sur la branche `dev` est operationnel, alors il peuvent l'envoyer sur la fameuse branche `master` et le tour et jouer !

Encore une fois c'est un travail d'équipe et le plus important sur GIT est de communiquer avec son équipe, eviter de supprimer la branche master si possible et bien reflechir sur le decoupage et la repartition des taches.

# Outro

Donc voila un petit peu ce que je vous voulais vous presenter aujourd'hui Il s'agissait d'une video d'introduction à GIT, j'espere qu'elle vous auras plut Bien evidemment pleins de points n'ont volontairement pas été abordé pour ne pas trop surcharger d'informations les débutants.

Mais pour les plus curieux, je vous laisserais quelques liens en descriptions pour completer ce que l'ont a vu aujourd'hui.

Si la video vous a plut je vous encourage à mettre un j'aime A vous abonnez a la chaine avec la petit cloche A mettre un commentaire si le coeur vous en dit

Et nous ont se dit à la prochaine, pour de nouvelles videos, c'etait Graven, ciao à toi et à la prochaine ! hé hé !