Git - Remote - FR

Avant de commencer nous te conseillons de faire la quête suivante :

[Git - Les fondamentaux](https://odyssey.wildcodeschool.com/adventures/227130)

[Git](https://odyssey.wildcodeschool.com/adventures/227130)

*Cette quête t'apprendra à utiliser Git avec un repository distant.*

Objectifs

•

Découvrir et comprendre la notion de repository distant.

•

*Câbler* un repository distant avec son repository local.

•

Connaître et comprendre les commandes de base liées au repo distant.

•

clone

•

remote

•

push

•

pull

•

fetch

•

merge

Challenge ️🕹️

Pour valider cette quête tu devras résoudre le challenge: *Envoi ton code sur github*. Le principe du challenge est détaillé dans l’onglet [Challenge](https://odyssey.wildcodeschool.com/quests/104/challenge).

[VOIR LE CHALLENGE](https://odyssey.wildcodeschool.com/quests/104/challenge)

Rappel

Afin de te remémorer un peu les commandes de la précédente quête "*Git - les fondamentaux*" et de te familiariser avec quelques-unes des nouvelles que tu vas découvrir ici,

voici une ressource interactive te permettant de tester diverses commandes usuelles.

[Try Git Challenge - En anglais](https://try.github.io/levels/1/challenges/1" \t "_blank)

[Suis les indications du site et saisis les commandes demandées.](https://try.github.io/levels/1/challenges/1" \t "_blank)



https://try.github.io/levels/1/challenges/1

Le repository distant

Jusqu'à présent tu as travaillé uniquement avec ton repository local.

C'est très pratique et cela permet de démarrer un projet rapidement ou de faire des tests facilement sans impacter tous tes collègues.

Mais à un moment il faut envoyer tes sources sur un repository distant afin de :

•

Sauvegarder tes modifications sur un serveur.

•

Mettre en commun ton code avec tes collègues.

Avant de voir comment on fait, il faut déjà bien comprendre ce qu'est le repository distant. On parle aussi de "remote".

Il existe plusieurs plateformes pour héberger tes sources distantes, en voici quelques-unes :

•

[GitHub - la plus connue](https://github.com/)

•

[Bitbucket](https://bitbucket.org/product)

•

[Gitlab](https://about.gitlab.com/)

Ou tu peux également créer ton propre serveur Git avec ou sans interface d'administration web (ça n'est pas le but de cette quête).

À la Wild Code School, nous utilisons GitHub mais lors de ton stage tu seras peut être sur une autre plateforme, on retrouve toujours plus ou moins les mêmes fonctionnalités.

Paramétrage du remote

Afin de pouvoir envoyer tes sources vers un repository distant, il faut commencer par informer ton repository local de l'existance du distant.

Pour cela il te faut renseigner un "remote", ton repository local peux contenir plusieurs remotes mais très souvent il n'en a qu'un.

Les remotes portent un nom qui permet de les manipuler, par convention le remote "de base" se nomme "origin"

Il existe plusieurs manières de "câbler" son remote.

Tu ne possèdes pas encore de repository distant (c'est notre cas).

En premier lieu tu dois créer ton repository distant (dans notre cas avec github).

Une fois les différentes étapes dans GitHub réalisées, tu dois copier l'url que tu peux trouver dans le "quick setup" puis taper la commande suivante dans ton terminal (dans le repertoire de ton projet).

git remote add origin https://github.com/{username}/{projectname}.git

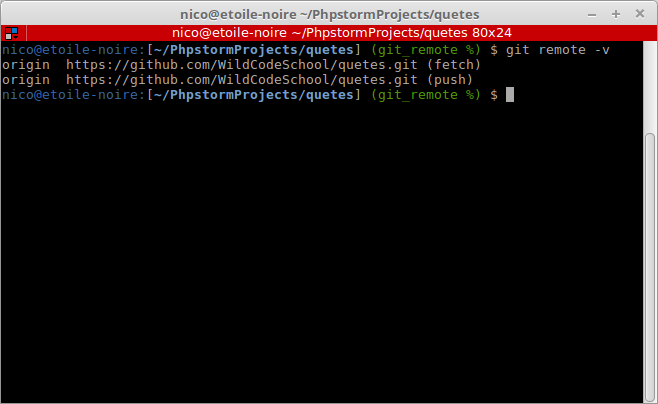
Shell

Dans cette commande, *origin* est le nom du remote, auquel on associe une URL. C'est un nom comme un autre, utilisé par convention, mais on aurait très bien pu en choisir un autre.

Ca y est, ton remote est paramétré (en fetch et en push). Tu peux confirmer cela en lançant la commande suivante à tout moment.

git remote -v

Shell



•

Fetch : Pour récupérer du distant vers le local.

•

Push : Pour envoyer du local vers le distant.

Tu possèdes déjà un repository distant (un projet déjà existant avec du code)

Dans ce cas ci, c'est plus simple. Il te suffit de récupérer le projet en local grâce à la commande "clone" :

git clone https://github.com/{username}/{projectname}.git

Shell

Ceci va récupérer le projet distant en local ET "câbler" correctement le remote en même temps.

Tu peux vérifier en faisant comme précédemment :

git remote -v

Shell

Tu peux donc utiliser l'une ou l'autre des méthodes expliquées ci-dessus mais ton projet local doit être correctement paramétré en fetch et en push.

Envoyer du code vers le respository distant

Pour envoyer du code vers le repository distant, tu devras utiliser la commande "push".

La commande push met à jour le repository distant avec le contenu des derniers commits locaux non envoyés.

Dans la quête précédente "*Git - les fondamentaux*" tu avais fait un commit sur ta branche *master*, puis au moins un commit sur ta branche nommée *wcs*.

Tu vas envoyer ces commits vers ton repository distant.

*Remarque :* ***à ce stade, tes repositories, local et distant, sont liés****... Mais il faut ensuite créer une branche distante pour chaque branche locale que tu veux "pousser".*

C'est ce qu'on va faire maintenant. A l'issue de la quête précédente, tu avais dû rester dans la branche *wcs*

(Logiquement ton terminal doit te l'indiquer mais tu peux le vérifier en faisant un git branch, la branche étoilée est ta branche actuelle).

Si c'est le cas, retourne sur ta branche *master* :

git checkout master

Shell

Puis pousse ton commit vers le repository :

git push -u origin master

Shell

Si on décompose cette commande, git push est la commande habituelle pour *pousser* vers le remote. L'option -u est importante car elle permet de lier ta branche locale à la branche distante. *origin* est le nom du remote, le même que celui donné comme paramètre à git remote add. *master* est le nom de la branche à créer sur le remote, le même que celui de ta branche locale actuelle.

*Remarque :* ***l'option -u n'est à utiliser que lors de ton premier push de cette branche***

En effet, après la commande que tu viens d'entrer, tes branches *master* locale et distante sont liées. Lors des prochains push depuis ta branche *master* locale, Git saura qu'il doit pousser vers la branche *master* distante.

Maintenant, retourne sur ta branche *wcs*, en faisant :

git checkout wcs

Shell

Puis pousse tes commits vers le repository

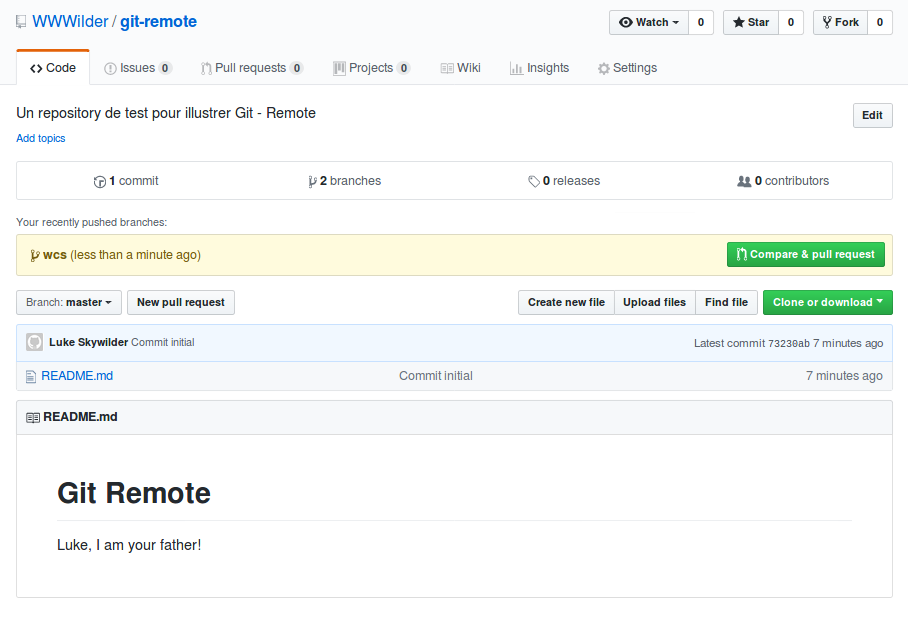
git push -u origin wcs

Shell

Comme c'est ton premier push depuis la branche *wcs*, il est à nouveau nécessaire de préciser la branche distante, juste pour ce push.

Il est possible d'utiliser le push de plusieurs façons en fonction des arguments, mais pour l'instant, retiens juste qu'on précise le remote et la branche la première fois, et que par la suite, git push suffit.

Maintenant, recharge la page de ton repository sur GitHub : il n'est plus vide ! Il contient deux branches, le fichier README.md que tu avais créé, et un commit.



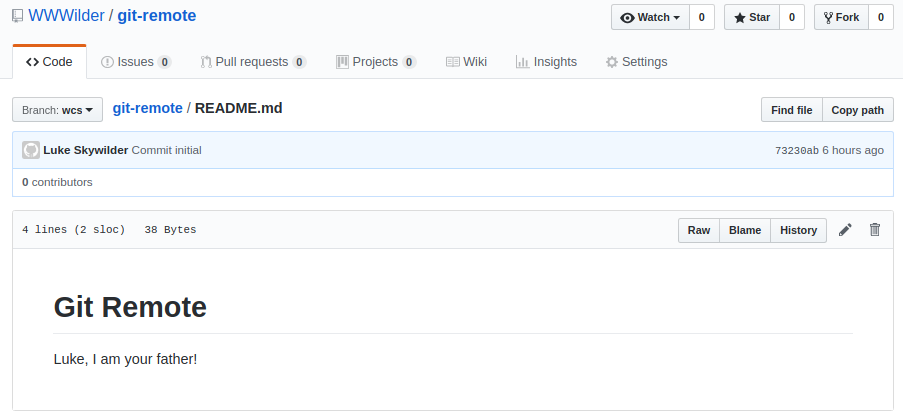
***Wait!*** *Pourquoi un fichier et un commit, alors que tu devrais logiquement trouver ton fichier wild.php et le(s) commit(s) correspondants ? C'est simple : par défaut, la page d'accueil d'un repository sur GitHub affiche l'état actuel (fichiers, nombre de commits) de la branche master.*

Récupérer du code du repository distant

Dans le cas de la quête tu es seul à travailler sur ton projet mais ça sera rarement le cas.

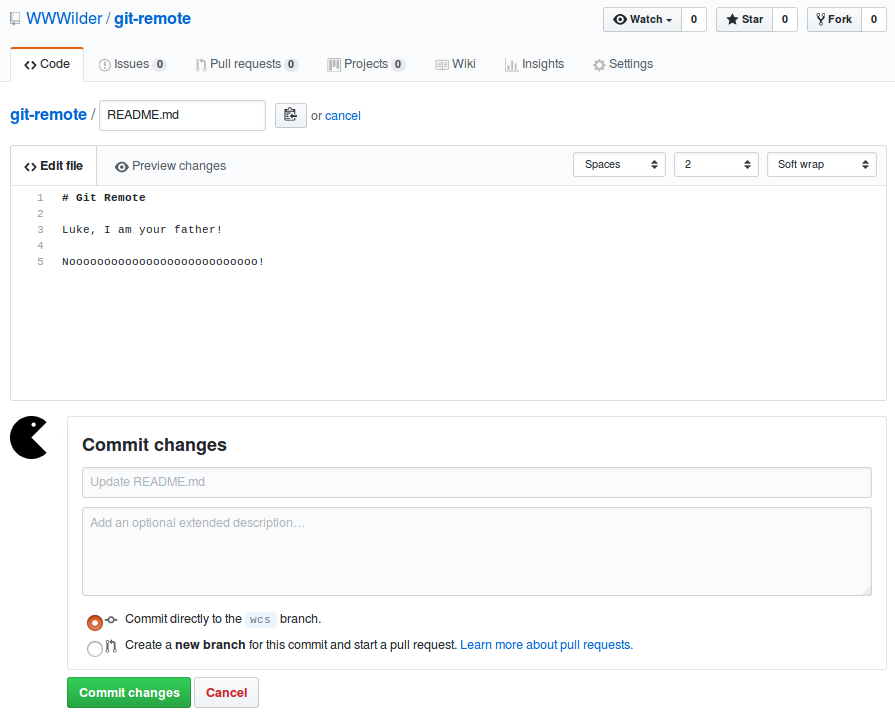
Tu vas donc simuler une modification faite sur le repository distant.

Pour cela, depuis la page du repo sur GitHub, clique sur "2 branches", puis dans la liste, clique sur "wcs". Cette fois tes deux fichiers README.md et wild.php s'affichent. Clique sur le fichier README.md, tu devrais voir ceci :



Clique sur le petit crayon pour éditer le fichier, fais une modification dedans.

Tout en bas de l'écran tu devrais voir ceci :



Clic sur le bouton vert "Commit changes" en vérifiant bien que le radio bouton est sur "Commit directly to the wcs branch".

Tu viens de faire une modification du fichier README.md qui a été intégrée dans un nouveau commit : on parle parfois de révision de fichier.

Cette modification, pour le moment, n'est **QUE SUR GitHub (pas en local)**. Comme si un autre développeur avait "pushé" des modifications sur ta branche wcs.

Tu vas donc devoir récupérer cette modification sur ton repository local en lançant la commande :

git pull origin wcs

Shell

Ceci est un raccourci, le *pull* en réalité fait d'abord un fetch puis un merge (et il existe d'autres manières de le paramétrer). Donc équivalent à :

git fetch origin

git merge origin/wcs

Shell

La commande "fetch" récupère en local tous les "objets git" d'un remote.

La commande "merge" fusionne 2 branches ensemble, la branche sur laquelle on se trouve avec celle donnée en paramètre.



Il faut bien comprendre qu'en local il existe une copie de la branche distante ainsi qu'une branche uniquement locale.

Lorsque tu lances la commande (à partir de la branche wcs)

git merge origin/wcs

Shell

Git ne travaille qu'en local donc tu lui demandes de fusionner la branche distante nommée "wcs" située sur le remote "origin" dans la branche locale sur laquelle tu te trouves (wcs).

D'où l'importance d'avoir fait le fetch avant puisque *fetch* met à jour ta branche distante en local (origin/wcs).

Une fois ceci fait, ton projet local est à jour avec les modifications de tes collègues.

[Git Fetch Pull Clone Rebase](https://blog.mikepearce.net/2010/05/18/the-difference-between-git-pull-git-fetch-and-git-clone-and-git-rebase/" \t "_blank)

[Article récapitulatif sur les commandes Pull, Fetch, Clone et Rebase (Rebase arrivera dans la prochaine quête).](https://blog.mikepearce.net/2010/05/18/the-difference-between-git-pull-git-fetch-and-git-clone-and-git-rebase/" \t "_blank)



https://blog.mikepearce.net/2010/05/18/the-difference-between-git-pull-git-fetch-and-git-clone-and-git-rebase/

[Git remote push pull](https://www.grafikart.fr/formations/git/remote-push-pull" \t "_blank)

[Une vidéo explicative qui reprend un peu tout ce qui concerne le "remote".  
Tu peux éventuellement sauter la partie ou il initialise le dossier serveur (git init --bare) et donc commencer à partir de 3'10 environ.](https://www.grafikart.fr/formations/git/remote-push-pull" \t "_blank)



https://www.grafikart.fr/formations/git/remote-push-pull

Challenge ️🕹️

**Envoi ton code sur github**

Énoncé :

Pour commencer fait un

git status

Shell

Afin de t'assurer que tu n'as pas de fichiers non "addés" (c'est à dire modifiés dans ton répertoire de travail mais pas dans l'index)

Si tu n'es pas à jour, ajoute les fichiers qui ne le seraient pas (et qui doivent l'être), puis commit tes modifications.

Ensuite, on suppose que ton travail dans la branche wcs est fini, testé et correct. Donc tu dois le réintégrer dans la branche master.

Pour cela, tu as plusieurs solutions mais pour cette quête nous allons utiliser un *merge* sans option particulière.

Retourne sur la branche master et mets-la à jour :

git checkout master

git pull origin master

Shell

Bon, la deuxième commande n'était pas vraiment utile ici, et doit se solder par le message "Already up-to-date.". Ceci dit, quand on travaille à plusieurs sur un projet, c'est toujours une bonne habitude de faire un "git pull" quand tu changes de branche, au cas où un de tes collègues ait poussé des modifications sur le remote.

*Question : connais-tu une autre façon de faire pour arriver au même résultat que le pull ?*

Maintenant tu dois "merger" la branche wcs dans master, pour cela :

git merge wcs

Shell

Dans le cas ou tu n'as que des ajouts (c'est le cas ici), tu n'auras aucun souci.

Par contre, si un fichier, présent dans la branche master, a été modifié dans la branche wcs, sur une ligne déjà existante, Git te demandera de régler le conflit manuellement.

Concrètement, il va te proposer les 2 versions, celle de la branche wcs et celle de la branche master, à toi de choisir laquelle est correcte (discute avec ton équipe car master a pu évoluer).

Rends-toi sur GitHub dans l'onglet "code" et soumet cette url dans Odyssey.

Aller plus loin avec Git

Avant de commencer nous te conseillons de faire la quête suivante :

[Git - Remote - FR](https://odyssey.wildcodeschool.com/adventures/229738)

[Git](https://odyssey.wildcodeschool.com/adventures/229738)

Jusqu'à présent, tout s'est bien passé pour toi avec Git et Github (je suppose...) mais si tout va mal, que faire ?!

Cette quête va répondre en partie à cette interrogation.

Dans un second temps, tu verras comment gérer des versions nommées de logiciel (tags)

Objectifs

•

Être capable de gérer des erreurs de manipulation et/ou conflits lors de la gestion locale

•

revert

•

reset

•

stash

•

checkout

•

rebase

•

Être capable de gérer les tags

•

tag

Challenge ️🕹️

Pour valider cette quête tu devras résoudre le challenge: *Tag ton projet*. Le principe du challenge est détaillé dans l’onglet [Challenge](https://odyssey.wildcodeschool.com/quests/115/challenge).

[VOIR LE CHALLENGE](https://odyssey.wildcodeschool.com/quests/115/challenge)

Revenir en arrière

Pour cette quête tu vas repartir du projet qui a servi lors des 2 précédentes quêtes sur Git :

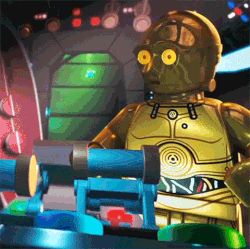
•

*Git fundamentals*

•

*Git Remote*

Si tu ne fais jamais d'erreur, tu peux sauter cette partie... Mais sinon attache ta ceinture, c'est parti !



Pour bien comprendre comment corriger une erreur, tu dois déjà savoir où se situe ton erreur. Pour rappel, tes fichiers passent par 3 "étapes" :

•

espace de travail (fichiers pas encore "addés")

•

Index (ou stage) (fichiers "addés" mais pas "commités")

•

HEAD (fichiers "commités")

Tu es dans le cas où tu sais que ton développement est buggé et tu as identifié un commit ou le projet fonctionnait bien.

Dans ce cas là, tu peux utiliser :

•

reset (hard ou soft)

•

revert

•

checkout

Voici quelques ressources qui vont te permettre de mieux comprendre ces 3 commandes.

[Git checkout revert reset](https://www.grafikart.fr/formations/git/checkout-revert-reset" \t "_blank)

[#checkout #revert #reset  
Une ressource vidéo de qualité.](https://www.grafikart.fr/formations/git/checkout-revert-reset" \t "_blank)



https://www.grafikart.fr/formations/git/checkout-revert-reset

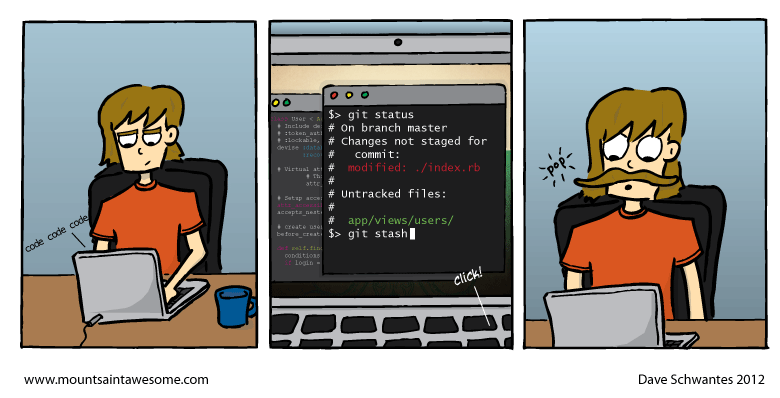
[Git undoing changes](https://www.atlassian.com/git/tutorials/undoing-changes" \t "_blank)

[#checkout #revert #reset](https://www.atlassian.com/git/tutorials/undoing-changes" \t "_blank)



https://www.atlassian.com/git/tutorials/undoing-changes

Stash - Le remisage



Lorsque tu veux faire un *merge* ou un *checkout* mais que tu as du travail non sauvegardé, par exemple si tu dois rapidement fixer un bug, Git t'empêchera de faire ces actions.

Tu as donc 2 solutions :

1.

Tout "adder" et "commiter" proprement (si possible)

2.

Stasher (remiser) ton travail.

La solution 1 ne nous intéresse pas car ton travail en cours n'est pas fini et donc pas "commitable".

Donc pour la solution 2, il suffit de lancer la commande :

git stash

Shell

Cela sauvegarde tes modifications dans un endroit spécial dédié aux stashs (tes modifications vont donc disparaître de ton espace de travail).

Pour restaurer tes modifications (celles que tu viens de stasher). Il suffit de lancer :

git stash apply

Shell

Mais tu pourrais avoir plusieurs stash donc pour voir la liste de tous les stash :

git stash list

Shell

et donc pour appliquer un stash bien précis que tu aurais repéré dans la liste :

git stash apply stash@{0}

Shell

Si tu veux faire le ménage, tu peux supprimer des stashs :

git stash drop stash@{0}

Shell

[Git stash, le remisage](https://www.grafikart.fr/formations/git/stash-remisage" \t "_blank)

[#stash #remisage](https://www.grafikart.fr/formations/git/stash-remisage" \t "_blank)



https://www.grafikart.fr/formations/git/stash-remisage

[Git stash - tutoriel](https://www.atlassian.com/git/tutorials/git-stash" \t "_blank)

[#stash #remisage](https://www.atlassian.com/git/tutorials/git-stash" \t "_blank)



https://www.atlassian.com/git/tutorials/git-stash

Les tags

Afin de retrouver une version bien particulière de ton projet, il est possible de les nommer via des tags.



La commande pour créer un tag est assez simple :

•

tag

Mais attention, elle ne crée le tag qu'en gitlocal, il faut bien penser à "pusher" vos tags vers votre repository distant.

Un simple push ne suffit pas, il faut faire un :

git push --tags

Shell

[Git tag](https://www.drupal.org/node/1066342" \t "_blank)



https://www.drupal.org/node/1066342