**GitHub Pour les Nuls : Pas de Panique, Lancez-Vous ! (Première Partie)**

**Un tutoriel en deux parties pour apprendre à démarrer sur GitHub et lancer votre premier projet.**

*Posté par Christophe Ducamp le dimanche 15 décembre 2013*

Traduction d’un article original de [Lauren Orsini](http://otakujournalist.com/about-the-author/) publié le 30 septembre 2013 pour ReadWriteWeb. Seul le [lien original fait référence](http://readwrite.com/2013/09/30/understanding-github-a-journey-for-beginners-part-1).

La traduction reste à raffiner avec la pratique de cet outil. [Seconde partie en cours d’étude](https://www.christopheducamp.com/2013/12/16/gitHub-pour-nuls-partie-2/) pour me lancer sous peu dans les premiers *commits* à la ligne de commande. Mise en forme prévue pour le plan de route [indieweb](http://indiewebamp.com/) 2014. Merci. - [xtof\_fr](https://twitter.com/xtof_fr/status/412238889739247616)

***GitHub est plus qu’un simple outil de programmation. Si vous voulez collaborer sur n’importe quoi, vous devriez l’essayer. 1ère Partie pour apprendre à démarrer sur GitHub.***

Nous sommes en 2013 et pas moyen d’y échapper : vous devrez apprendre comment utiliser GitHub.

Pourquoi ? Parce que c’est un réseau social qui change drastiquement notre façon de travailler. Ayant démarré sous forme de plateforme collaborative pour développeurs, GitHub est désormais le plus grand espace de stockage de travaux collaboratifs dans le monde. Que vous soyez intéressé(e) pour participer à ce cerveau global ou tout simplement pour partir à la recherche de cet énorme réservoir de connaissances, vous vous devez d’y être.

En étant simplement membre, vous pourrez croiser le fer avec ce qu’aiment [Google](https://github.com/google) et [Facebook](https://github.com/facebook). Avant que Github n’existe, les grandes sociétés créaient leurs bases de connaissance surtout en privé. Mais lorsque vous accédez à leurs comptes GitHub, vous êtes libres de télécharger, étudier et construire dessus tout ce que vous voulez sur ce qu’elles ajoutent sur ce réseau. Aussi, qu’attendez-vous ?

**Chercher des Réponses GitHub**

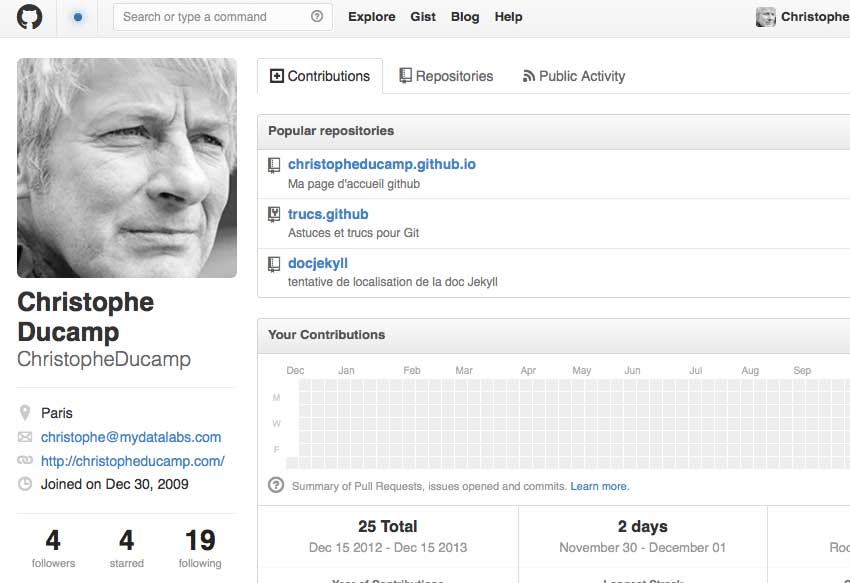
Aussi gênant que cela puisse paraître, j’ai écrit ce tutoriel parce que je me sentais vraiment perdue dans tous les articles de type “GitHub pour Débutants”. Probablement parce qu’à la différence de la plupart des utilisateurs de Github, je manque de bagage solide en programmation. Et je ne m’y retrouvais pas non plus dans les tutoriels d’utilisation de Github, pour construire une vitrine de quelques travaux de programmation.

Voir aussi : [Tom Preston-Werner de Github : Comment Nous Sommes Devenus Mainstream](http://readwrite.com/2013/11/18/github-tom-preston-warner)

Ce que vous pourriez ignorer, c’est qu’il existe plein de raisons d’utiliser GitHub même si vous n’êtes pas programmeur. Selon les vidéos de tutoriels GitHub, tout travailleur du savoir peut en tirer profit, “knowledge worker” s’entendant ici pour désigner la plupart des professionnels faisant usage d’un ordinateur.

Par conséquent, si vous avez déjà lâché prise sur la compréhension d’utilisation de Github, cet article est pour vous.

L’un des principaux malentendus concernant GitHub est que c’est un outil de développement, faisant partie de la panoplie de tout programmeur, comme le sont les langages de programmation et les compilateurs. Cependant, GitHub en lui-même n’est rien de plus qu’un réseau social comme Facebook ou Flickr. Vous construisez un profil, vous y déposez des projets à partager et vous vous connectez avec d’autres utilisateurs en suivant leurs comptes. Même si la plupart des utilisateurs y déposent des projets de programmes ou de code, rien ne vous empêche d’y placer des textes ou tout type de fichier à présenter dans vos répertoires de projets.



Vous avec peut-être déjà plus d’une dizaine de comptes sociaux… et voici pourquoi vous devriez être sur Github : il dispose des meilleures Conditions Générales d’Utilisation. Si vous regardez la section F des [conditions générales](https://help.github.com/articles/github-terms-of-service), vous verrez que Github fait tout pour vous assurer que vous conservez la propriété complète de tous les projets que vous déposez sur le site :

*We claim no intellectual property rights over the material you provide to the Service. Your profile and materials uploaded remain yours.*[*conditions générales GitHub*](https://help.github.com/articles/github-terms-of-service)

En outre, vous pouvez véritablement utiliser GitHub sans connaître UNE SEULE LIGNE de code. Vous n’avez pas besoin de tutoriel pour vous enregistrer et vous promener. Mais mon point de vue est que GitHub a le mérite de nous apprendre à faire les choses dures en premier, ce qui veut dire, utiliser le bon vieux code Git. Après tout, GitHub est parvenu à produire l’une des interfaces graphiques sans effort pour le langage de programmation Git.

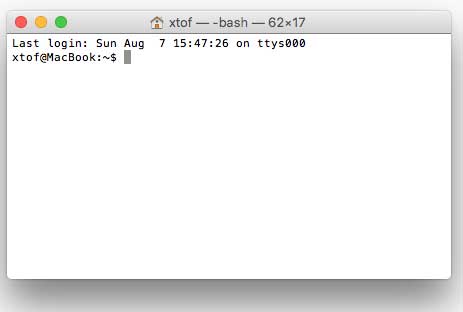
**C’est Quoi Git ?**

Remercions le célèbre développeur de logiciel [Linus Torvalds](http://fr.wikipedia.org/wiki/Linus_Torvalds) pour Git, le logiciel qui fait tourner le coeur de GitHub. (Et tant que vous y êtes, remercions-le aussi pour le système d’exploitation Linux). **Git est un logiciel de contrôle de version**, ce qui signifie qu’il gère les modifications d’un projet sans écraser n’importe quelle partie du projet. Et il ne risque pas de disparaître, tout particulièrement parce que Torvalds et ses collègues développeurs du noyau utilisent Git pour aider à développer le noyau coeur de Linux.

Pourquoi utiliser quelque chose comme Git ? Supposons que vous mettiez à jour avec un collègue des pages sur le même site web. Vous faites des modifications, vous les sauvegardez et les versez sur le site. À ce stade, tout va bien. Le problème survient quand votre collègue travaille sur la même page que vous en même temps. L’un de vous va voir son travail écrasé.

Une [application de contrôle de version](http://git-scm.com/video/what-is-version-control) comme Git empêche ça d’arriver. Vous et votre collègue pouvez chacun de votre côté verser vos révisions sur la même page, et Git sauvegardera deux copies. Plus tard, vous pourrez fusionner vos modifications sans perdre le travail dans le processus. Vous pouvez même revenir en arrière à tout moment, parce que Git conserve une “copie instantanée” de tous les changements produits.

Le problème avec Git est qu’il est vieux. Si vieux que nous devons utiliser la ligne de commande -ou l’application Terminal si vous êtes sur Mac - afin d’y accéder, et y taper dedans des bouts de code comme les hackers des années 90. Ceci peut être une proposition difficile pour les utilisateurs d’ordinateurs modernes. C’est là où Github entre dans la danse.



GitHub facilite l’utilisation de Git sur deux points. Premièrement, si vous [téléchargez le logiciel GitHub](https://desktop.github.com/) sur votre ordinateur, GitHub fournit une interface visuelle pour vous aider à gérer localement vos projets avec les contrôles de version. Deuxièmement, créer un compte sur GitHub.com apporte les contrôles de versions à vos projets web, et leur confère des fonctionnalités de réseaux sociaux.

Vous pouvez parcourir les projets d’autres utilisateurs de Github, et même y télécharger des copies pour vous-même afin de les modifier, apprendre ou les enrichir. D’autres utilisateurs peuvent faire la même chose avec vos projets publics, repérer vos erreurs et suggérer des corrections. De toute façon, aucune donnée ne se perd parce que Git enregistre un “instantané” de chaque modification.

Bien qu’il soit possible d’utiliser GitHub sans apprendre Git, il y a une énorme différence entre l’utilisation et la compréhension. Avant de connaître Git, je savais utiliser GitHub, mais je ne comprenais pas vraiment pourquoi. Dans ce tutoriel, nous allons apprendre à utiliser Git à la ligne de commande.

**Les Mots que les Personnes Utilisent quand Elles Parlent de Git**

Dans ce tutoriel, il y a quelques mots que j’utiliserai à plusieurs reprises, aucun d’eux dont je n’avais entendu parler avant d’avoir démarré l’apprentissage. Voici les plus connus :

**Ligne de Commande** : Le programme de l’ordinateur que nous utilisons pour entrer des commandes Git. Sur un Mac, ça s’appelle Terminal. Sur un PC, c’est un programme non-natif que vous téléchargez lorsque vous téléchargez Git pour la première fois (nous allon faire ça dans la section suivante). Dans les deux cas, vous tapez à l’écran des commandes à base de texte, appelées invites de commande, au lieu d’utiliser une souris.

**Dépôt :** Un répertoire ou de l’espace de stockage où vos projets peuvent vivre. Parfois les utilisateurs GitHub raccourcissent ça en “repo”. Il peut être local sur un répertoire de votre ordinateur, ou ce peut être un espace de stockage sur GitHub ou un autre hébergeur en ligne. À l’intérieur d’un dépôt, Vous pouvez conserver des fichiers de code, des fichiers texte, des images.

**Contrôle de Version :** Fondamentalement, l’objectif pour lequel Git a été conçu. Quand vous avez un fichier Microsoft Word, vous l’écrasez à chaque fois que vous faites une nouvelle sauvegarde, ou vous sauvegardez plusieurs versions. Avec Git, vous n’êtes plus obligé de faire ça. Git conserve des “instantanés” de chaque point dans l’historique d’un projet, de sorte que vous ne pouvez jamais le perdre ou l’écraser.

**Commit :** C’est la commande qui donne à Git toute sa puissance. Quand vous “committez”, vous prenez un “instantané”, une “photo” de votre dépôt à ce stade, vous donnant un point de contrôle que vous pouvez ensuite réévaluer ou restaurer votre projet à un état précédent.

**Branche :** Comment plusieurs personnes travaillant sur un projet en même temps sans que Git ne s’embrouille ? Habituellement, elles se “débranchent” du projet principal avec leurs propres versions complètes des modifications qu’elles ont chacune produites de leur côté. Après avoir terminé, il est temps de “fusionner” cette branche pour la ramener vers la branche “master”, le répertoire principal du projet.

**Commandes Spécifiques Git**

Le fait que Git ait été conçu avec un grand projet comme Linux, il existe beaucoup de commandes Git. Toutefois, pour utiliser les bases de Git, vous aurez seulemenet besoin de connaître quelques termes. Ils commencent tous de la même façon avec le mot “git”.

git init : Initialise un nouveau dépôt Git. Jusqu’à ce que vous exécutiez cette commande dans un dépôt ou répertoire, c’est juste un dossier ordinaire. Seulement après avoir entré cette commande, il accepte les commandes Git qui suivent.

git config : raccourci de “configurer,” ceci est tout particulièrement utile quand vous paramétrez Git pour la première fois.

git help : Oublié une commande ? Tapez-ça dans la ligne de commande pour afficher les 21 commandes les plus courantes de Git. Vous pouvez aussi être plus spécifique et saisir “git help init” ou tout autre terme pour voir comment utiliser et configurer une commande spécifique git.

git status : Vérifie le statut de votre repository. Voir quels fichiers sont à l’intérieur, quelles sont les modifications à *commiter*, et sur quelle branche du repository vous êtes en train de travailler.

git add : Ceci n’ajoute *pas* de nouveaux fichiers dans votre repository. Au lieu de cela, cela porte de nouveaux fichiers à l’attention de Git. Après avoir ajouté des fichiers, ils sont inclus dans les “instantanés” du dépôt Git.

git commit : la commande la plus importante de Git. Après avoir effectué toute sorte de modification, vous entrez ça afin de prendre un “instantané” du dépôt. Généralement cela s’écrit sous la forme git commit -m “Message ici“. Le -m indique que la section suivante de la commande devrait être lue comme un message.

git branch : Vous travaillez avec plusieurs collaborateurs et vous voulez produire des modifications de votre côté ? Cette commande vous permet de construire une nouvelle branche, ou une chronologie des commits, des modifications et des ajouts de fichiers qui sont complètement les vôtres. Votre titre va après la commande. Si vous vouliez créer une nouvelle branche appelée “chats”, vous saisiriez git branch chats.

git checkout : Permet littéralement de vérifier un dépôt dans lequel vous n’êtes pas. C’est une commande de navigation qui vous permet de vous déplacer vers le répertoire que vous voulez vérifier. Vous pouvez utiliser cette commande sous la forme git checkout master pour regarder la branche master, ou git checkout chats pour regarder une autre branche.

git merge : Lorsque vous avez fini de travailler sur une branche, vous pouvez fusionner vos modifications vers la branche master, qui est visible pour tous les collaborateurs. git merge chats prendrait toutes les modifications que vous avez apportées à la branche “cats” et les ajoutera à la la branche master.

git push : Si vous travaillez sur votre ordinateur local, et voulez que vos commits soient visibles aussi en ligne sur Github, vous “push”ez les modifications vers Github avec cette commande.

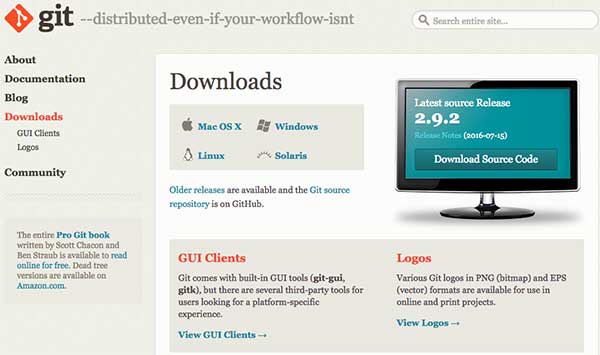
git pull : Si vous travaillez sur votre ordinateur local, et que vous voulez la version la plus à jour de votre repository pour travailler dessus, vous “pull”ez (tirez) les modifications provenant de Github avec cette commande.

**Paramétrer GitHub ET Git Pour La Première Fois**



Premièrement, vous devrez [vous enregistrer](https://github.com/) pour disposer d’un compte sur GitHub.com. C’est aussi simple que de s’enregistrer sur n’importe quel autre réseau social. Conservez l’e-mail que vous avez choisi à portée de main ; nous en aurons besoin de nouveau.

Vous pourriez vous arrêter là et github fonctionnerait bien. Mais si vous voulez travailler sur votre projet sur votre ordinateur local, vous devez avoir installé Git. En fait, Github ne fonctionnera pas sur votre ordinateur local si vous n’installez pas Git. [Téléchargez et installez la dernière version de Git pour Windows, Mac ou Linux selon votre machine](http://git-scm.com/downloads).



Maintenant, il est temps de passer à la ligne de commande. Sur Windows, ça veut dire démarrer l’application Git Bash que vous venez d’installer, et sur MacOSX, c’est le bon vieux Terminal. Il est temps de vous présenter à Git. Saisissez le code qui suit :

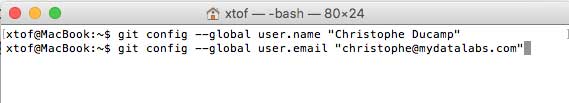
git config --global user.name "Votre Nom Ici"

Vous aurez bien sûr besoin de remplacer “Votre Nom Ici” par votre propre nom entre guillemets. Ce peut être votre nom légal, votre pseudo en ligne ou tout ce que vous voudrez. Git s’en moque, il a juste besoin de savoir à qui créditer les commits et projets futurs.

Ensuite, indiquez-lui votre adresse de courrier électronique et assurez-vous que c’est le même email que vous avez utilisé pour enregistrer votre compte Github. Faites comme suit :

git config --global user.email "votre\_email@votre\_email.com"

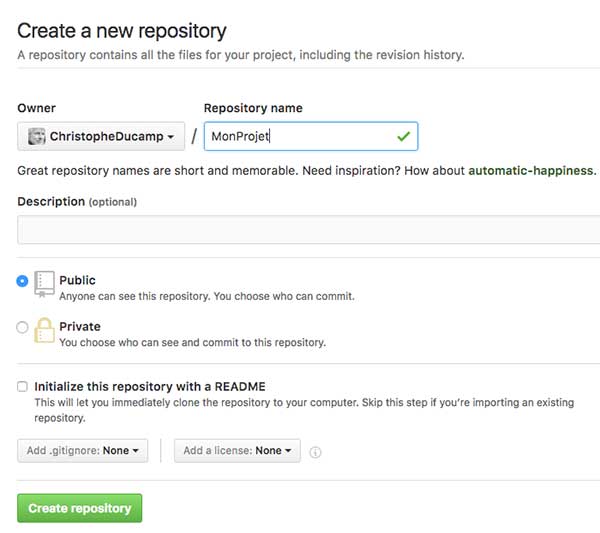
C’est tout ce que vous devez faire pour commencer à utiliser Git sur votre ordinateur. Cependant, puisque vous venez de configurer un compte Github.com, il est probable que vous ne souhaitiez pas gérer simplement votre projet localement, mais aussi en ligne. Si vous le souhaitez, vous pouvez également configurer Git pour qu’il ne vous demande pas de vous connecter à votre compte Github à chaque fois que vous voulez lui parler. Pour les besoins de ce tutoriel, ce n’est pas un grand problème parce que nous ne lui parlerons qu’une fois. Le tutoriel complet pour faire ça est [situé sur Github](https://help.github.com/articles/set-up-git).



**Créer Votre Repo En Ligne**

Maintenant que vous avez tout mis en place, il est temps de créer un endroit pour placer votre projet à faire vivre. Git et Github appellent cela un repository ou “repo” pour faire court, un répertoire numérique ou un espace de stockage où vous pouvez accéder à votre projet, ses fichiers, et toutes les versions de ses fichiers que Git sauvegarde.

Retournons sur GitHub.com et cliquez sur la petite icône de texte à côté de votre nom d’utilisateur. Ou allez vers la nouvelle page repository si toutes les icônes sont les mêmes. Donnez à votre dépôt un nom court et mémorisable. Allez-y et rendez-le public, pourquoi cacher votre tentative d’apprendre Github !

.

Ne vous inquiétez pas de cliquer sur le bouton radio à côté intitulé “Initialize this repository with a README.” Un fichier Readme est généralement un fichier texte qui explique sommairement le projet. Mais nous pouvons produire localement notre propre fichier Readme pour l’entraînement.

Cliquez sur le bouton vert “Create Repository” et c’est fait. Vous avez maintenant un espace en ligne pour votre projet restant à faire vivre.

**Créer Votre Repository Local**

Ainsi nous venons juste de créer un espace pour votre projet en ligne, mais ce n’est pas l’endroit où nous travaillerons dessus. La majeure partie de votre travail sera faite sur votre ordinateur. Nous devons donc en fait refléter ce repository que nous venons juste de produire sous un répertoire local.

C’est –là où nous faisons quelques saisies de ligne de commande– la partie de chaque tutoriel Git qui me chahute le plus, aussi j’irai vraiment lentement.

Saisissez d’abord :

mkdir ~/MonProjet

mkdir est le raccourci de “make directory”. Ce n’est en réalité pas une ligne de commande Git, mais une commande générale de navigation provenant du temps avant les interfaces ordinateurs visuelles. Le ~/ veille à vous assurer de construire le repository au niveau supérieur de la structure de fichiers de notre ordinateur, au lieu d’un répertoire coincé dans quelque autre répertoire qui serait plus difficile à retrouver. En fait, si vous saisissez ~/ dans la fenêtre de votre navigateur, cela vous ramènera vers le répertoire local le plus haut de votre ordinateur. Pour moi, en utilisant Chrome sur un Mac, cela affiche mon dossier Users.

Remarquez aussi que je l’ai appelé MonProjet, le même nom donné au dépôt Github que nous avons produit précédemment. Assurez-vous aussi de garder une cohérence sur votre nom.

Puis, saisissez :

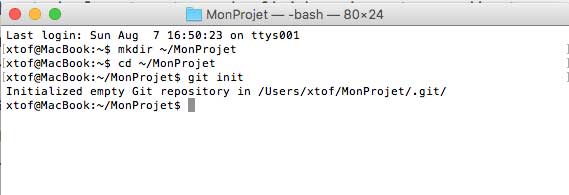
cd ~/MonProjet

cd signifie change directory, et c’est aussi une commande de navigation. Nous venons de produire un répertoire, et nous voulons maintenant passer à ce répertoire et aller dedans. Une fois que nous avons tapé cettte commande, nous sommes transportés à l’intérieur de MonProjet.

Maintenant, nous allons pour finir utiliser une commande Git. Pour votre prochaine ligne, saisissez :

git init

Vous savez que vous utilisez une commande Git car elle démarre toujours par git. Init signifie “initialiser”. Souvenez-vous comment les deux précédentes commandes que nous avons saisies étaient des termes généraux de ligne de commande ? Quand nous tapons ce code à l’intérieur, cela dit à l’ordinateur de reconnaître ce répertoire comme un dépôt local Git. Si vous ouvrez le répertoire, il ne s’affichera pas différemment, parce que ce nouveau répertoire Git est un fichier caché à l’intérieur du dépôt dédié.



Cependant, votre ordinateur réalise maintenant que ce répertoire est *prêt-pour-git*, et vous pouvez commencer à entrer des commandes Git. Maintenant, vous avez à la fois un dépôt local et un repo en ligne pour votre projet. Dans la [seconde partie](https://www.christopheducamp.com/2013/12/16/gitHub-pour-nuls-partie-2/), vous apprendrez comment faire votre premier commit vers des dépôts locaux et sur Github. Et vous en saurez plus sur quelques ressources géniales de Github.

**GitHub pour les Débutants : Consignez, Poussez et Foncez !**

**Tutoriel pour devenir un utilisateur officiel de git !**

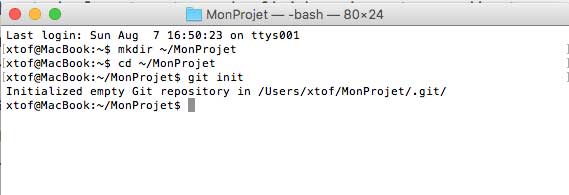
*Posté par Christophe Ducamp le lundi 16 décembre 2013*

*Traduction -à des fins d’étude et de mémo- d’un article original de*[*Lauren Orsini*](http://otakujournalist.com/about-the-author/)*publié le 2 octobre 2013 pour ReadWriteWeb. Seul le*[*lien original fait référence*](http://readwrite.com/2013/10/02/github-for-beginners-part-2)*. - [xtof\_fr](https://twitter.com/xtof_fr/status/412363441056129024)*

***Maintenant que nous connaissons les concepts Git, il est temps de jouer. Voici venue la deuxième partie de notre série.***

Dans la [1ère partie de ce tutoriel GitHub en deux parties](https://www.christopheducamp.com/2013/12/15/github-pour-nuls-partie-1/), nous avons examiné les principales utilisations de GitHub, commencé le processus d’enregistrement d’un compte GitHub et pour finir, nous avons créé notre propre repository local pour le code.

Maintenant que ces premières étapes ont été accomplies, ajoutons la première partie de votre projet en **produisant votre premier “commit” sur GitHub.**Lorsque nous nous sommes quittés dans la première partie, nous avions créé un repo local appelé MonProjet, qui, vu à la ligne de commande, ressemble à cette capture d’écran :



Toujours en fenêtre Terminal, à la prochaine ligne, entrez :

$ touch Readme.txt

Une fois de plus, *ceci n’est pas* une commande Git. C’est simplement une autre invite de commande standard de navigation. touch signifie en fait “créer”. Tout ce que vous écrivez après ce qui suit, c’est le nom de la chose créée. Si vous allez sur votre répertoire local en utilisant le Finder ou le menu Démarrer, vous verrez qu’un fichier vide intitulé Readme.txt se niche désormais dedans. Vous auriez pu varier le plaisir avec quelque chose comme “Lisez-moi.doc” ou “Kiwi.gif,” .

Vous pouvez clairement voir votre nouveau fichier Readme. Mais Git le peut-il aussi ? Regardons ça. Tapez :

$ git status

La ligne de commande, généralement passive jusqu’à ce stade, vous renvoie quelques lignes de texte similaires à ce qui suit :

*# On branch master*

*#*

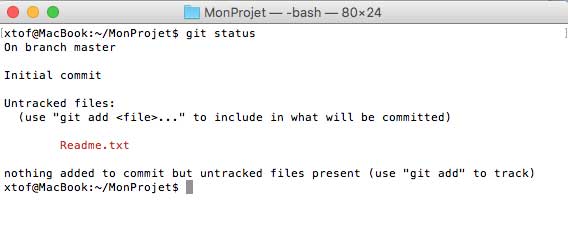
*# Untracked files:*

*#*

*# (use "git add ..." to include in what will be committed)*

*#*

*# Readme.txt*



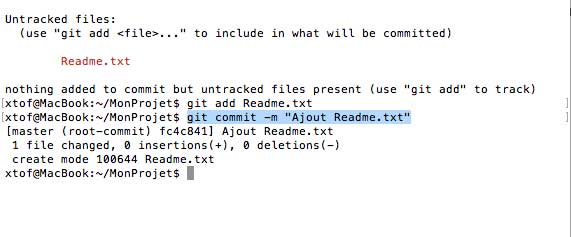
Qu’est-ce qui se passe ?

Tout d’abord, vous êtes sur la branche master, la branche principale de votre projet, ce qui est logique puisque nous n’avons pas “bifurqué” du projet. Il n’y a aucune raison faire ça puisque nous travaillons seul. Deuxièmement, Readme.txtest répertoriré comme un fichier “untracked”, ce qui signifie que Git l’ignore pour l’instant. Pour signaler à Git que le fichier est là, tapez :

$ git add Readme.txt

Remarquez comment la ligne de commande vous glisse un truc ici ? Très bien, nous avons ajouté notre premier fichier, aussi est-il temps à ce stade de prendre un “instantané” du projet, ou de le “consigner” :

$ git commit -m "Ajout Lisez-moi.txt"



Le marqueur m comme indiqué dans le répertoire des définitions en [1ère partie](https://www.christopheducamp.com/2013/12/15/Github-pour-nuls-partie-1/), indique simplement que le texte qui suit doit être lu comme un message. Notez que le message de commit est écrit au présent. Vous devriez **toujours écrire vos commandes au temps présent** parce que le contrôle de version ne traite que de flexibilité dans le temps. Vous n’écrivez pas pour dire ce qu’un commit a fait précédemment, parce que vous pouvez toujours revenir à la version précédente. **Vous écrivez ce que fait un commit**.

Maintenant que nous avons fait un petit travail en local, il est temps de pousser (de “push”er) notre premier “commit” sur GitHub.

“Attendez, on n’a jamais connecté mon dépôt en ligne à mon dépôt local,” pourriez-vous penser. Et vous avez raison. En fait, votre dépôt local et votre dépôt en ligne ne communiquent que par de courtes rafales, lorsque vous confirmez les ajouts et les modifications au projet. Passons maintenant à votre première véritable connexion.

**Connecter Votre Dépôt Local À Votre Dépôt GitHub**

Avoir à la fois un dépôt local et un dépôt à distance (en ligne), c’est tout bonnement le meilleur des deux mondes. Vous pouvez bricoler tout ce que vous aimez sans même être connecté à internet, tout en présentant votre travail fini sur Github afin que tout le monde puisse le voir.

Cette configuration facilite aussi le fait d’avoir plusieurs collaborateurs travaillant sur le même projet. Chacun de vous peut travailler seul sur son propre ordinateur, mais téléverser ou “push”er vos modifications vers le dépôt Github quand elles sont prêtes. Aussi allons-y.

Tout d’abord, nous devons dire à Git qu’un dépôt distant existe quelque part en ligne. Nous faisons ça en ajoutant ça à la connaissance de Git. Tout comme Git ne reconnaît pas nos fichiers jusqu’à ce que nous utilisions la commande git add, il ne reconnaîtra pas non plus notre dépôt distant à cette heure.

Supposons que nous ayons un dépôt GitHub appelé “MonProjet” situé sur https://github.com/nomutilisateur/MonProjet.git. Bien sûr, nomutilisateurdevrait être remplacé par votre véritable nom d’utilisateur Github, et MonProjetdevrait être remplacé par le véritable titre que vous avez donné à votre premier dépôt GitHub.

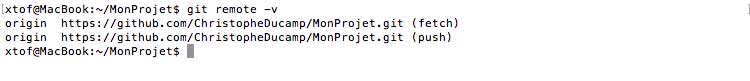
$ git remote add origin https://github.com/nomutilisateur/MonProjet.git

image

La première partie est connue ; nous avons déjà utilisé git add avec les fichiers. Nous avons ajouté après le mot origin pour indiquer un nouvel endroit à partir duquel viendront les fichiers. remote est un descripteur de origin, pour indiquer que l’original n’est *pas* sur l’ordinateur, mais quelque part en ligne.

Git sait désormais qu’il existe un dépôt distant et que c’est là où vous voulez envoyer vos modifications du dépôt local. Pour confirmer, saisissez cela pour déposer :

$ git remote -v

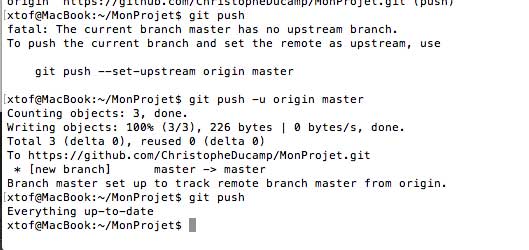


Cette commande vous donne une liste de toutes les origines distantes connues par votre dépôt local. En supposant que vous m’ayez suivi jusque là, il ne devrait y en avoir qu’un, le MonProjet.git que nous venons d’ajouter. Il est listé deux fois, ce qui signifie qu’il est disponible pour y *“push”*er de l’information, et pour y extraire (fetch) de l’information.

Maintenant, nous voulons téléverser, ou “push“er, nos modifications vers le dépôt distant Github. C’est facile. Tapez simplement :

$ git push

La ligne de commande vous soufflera plusieurs lignes, et le mot final qu’elle recrachera ressemblera à quelque chose comme “everything up-to-date.”



Git m’a renvoyé dans mon cas un paquet d’avertissements parce que j’avais simplement produit la commande simple. Pour être plus spécifique, j’ai saisi git push --set-upstream origin master, pour spécifier que je voulais dire la branche master de mon dépôt.

Connectez-vous de nouveau à GitHub. Vous remarquerez que GitHub suit désormais combien de commits vous avez produits aujourd’hui. Si vous avez suivi simplement ce tutoriel, ceci devrait être exactement un. Cliquez sur votre dépôt et il aura un fichier identique Readme.txt car nous l’avons construit précédemment à l’intérieur de votre dépôt local.

**Tous ensemble Maintenant !**

Bravo, vous êtes officiellement un utilisateur Git ! Vous pouvez créer des dépôts et *committer* des modifications. C’est là où s’arrête ce tutoriel de débutant.

*Regardez aussi :*[*Tom Preston-Werner de Github : How We Went Mainstream*](http://readwrite.com/2013/11/18/github-tom-preston-warner)

Cependant, vous pouvez avoir cette lancinante impression de ne pas vous sentir comme un expert. Bien sûr, vous avez réussi à suivre quelques étapes, mais êtes-vous prêt à y aller seul ? Je n’aurai nullement cette prétention.

Afin d’être plus à l’aise avec Git, avançons sur un workflow imaginaire tout en utilisant les quelques points que nous avons appris. Vous êtes désormais salarié dans l’agence “123 Web Design”, où vous construisez un nouveau site web pour le Magasin de Glaces de Jimmy avec quelques-uns de vos collègues.

Vous étiez un peu nerveux quand votre patron vous a demandé de participer au projet de redesign de refonte du site du Magasin de Glaces de Jimmy. Après tout, vous n’êtes pas programmeur ; vous êtes designer graphique. Mais votre patron vous a assuré que tout le monde peut utiliser Git.

Vous avez créé quelques nouvelles illustrations d’un sundae à la crème et il est temps de les ajouter au projet. Vous les avez enregistrées dans un dossier de votre ordinateur, appelé “icecream”, pour éviter de vous emmêler.

Ouvrez la Ligne de Commande et changez le répertoire jusqu’à ce vous soyez dans le répertoire icecream, là où est stocké votre design.

$ cd ~/icecream

Puis, réinitialisez Git de manière à pouvoir démarrer en utilisant des commandes Git à l’intérieur du répertoire. Le dossier est désormais un dépôt Git.

$ git init

Attendez, est-ce le bon fichier ? Voici comment vérifier et vous assurer que c’est bien l’endroit où vous avez stocké votre design :

$ git status

Et c’est ce que Git vous dira en retour :

*# Untracked files:*

*# (use "git add ..." to include in what will be committed)*

*#*

*# chocolat.jpeg*

Ayé ils sont ici ! Ajoutez-les dans votre dépôt local Git pour qu’ils soient suivis par Git.

$ git add chocolat.jpeg

Maintenant, faites un “instantané” du dépôt tel qu’il est maintenant avec la commande commit :

$ git commit -m "Ajoute chocolat.jpeg."

Bravo ! Mais vos collègues, acharnés au boulot dans leurs propres dépôts locaux, ne peuvent pas voir votre tout nouveau design ! Ceci parce que le projet principal est stocké dans le compte Github de la société (nom d’utilisateur : 123WebDesign) dans le dépôt intitulé “icecream.”

Parce que vous ne vous êtes pas encore connecté au dépôt GitHub, votre ordinateur ne sait même pas qu’il existe. Aussi, déclarez votre dépôt local :

$ git remote add origin https://github.com/123WebDesign/icecream.git

Et double-checkez pour vous assurer qu’il le connaît :

$ git remote -v

Pour finir, c’est le moment que vous attendiez. Téléversez ce délicieux sundae sur le projet :

git push

Tut tut ! Avec tous ces outils à portée de mains, il est clair que Git et le service GitHub ne sont pas que pour les programmeurs.

**Les Ressources Git**



Git est dense, je sais. J’ai fait de mon mieux pour produire un tutoriel qui pourrait même m’aider à savoir comment l’utiliser, mais nous n’apprenons pas tous de la même manière.

En plus de mon [anti-sèche pour la ligne de commande](https://www.christopheducamp.com/2013/12/09/anti-seche-ligne-de-commande/), voici quelques ressources que j’ai trouvées utiles tout en apprenant personnellement à utiliser Git et Github durant l’été :

* [**Pro Git**](http://git-scm.com/book/fr). Voici un livre complet open source sur l’apprentissage et l’utilisation de Git. Il peut paraître long, mais je n’ai pas eu besoin de lire quoi que ce soit après le chapitre trois pour apprendre les fondamentaux.
* [**Try Git**](http://www.codeschool.com/courses/try-git). CodeSchool et GitHub ont fait équipe pour produire ce tutoriel rapide. Si vous voulez un peu plus de pratique avec les fondamentaux, ceci devrait vous aider. Et si vous avez un peu d’argent en plus et que vous vouliez apprendre tout ce qu’il faut savoir sur Git, Git Real de Code School devrait faire l’affaire.
* [**GitHub Guides**](http://www.youtube.com/GitHubGuides). Si vous êtes plutôt visuel, le canal officiel de GitHub vaut le coup d’oeil. J’ai particulièrement beaucoup **appris de la série**[**Git Basics**](http://www.youtube.com/watch?v=8oRjP8yj2Wo&list=PLg7s6cbtAD165JTRsXh8ofwRw0PqUnkVH) en quatre parties.
* [**Git Reference**](http://gitref.org/). Vous avez les basiques mais vous oubliez toujours les commandes ? Ce site pratique est génial comme glossaire de référence.
* [**Git - le petit guide**](http://rogerdudler.github.io/git-guide/index.fr.html). Ce tutoriel est court et délicieux, mais il allait un peu trop vite pour moi. Si vous voulez vous rafraîchir sur les fondamentaux de Git, ceci devrait faire tout ce dont vous avez besoin.

**Glossaire Git pour les Nuls**

**git init** : Initialise un dépôt git dans un répertoire vide.

**git status** : Affiche le statut des choses dans le répertoire suivi

**git add <NOM DU FICHIER A SUIVRE>** : Ajoute un fichier à suivre dans la zone d’attente

**git commit -m "entrez votre message"** : Consigner (déclarer) toute modification.

**git remote add origin [adresse URL GitHub ici]** : Connecte votre dépôt local avec GitHub.

**git push -u origin nom\_branche** : Pousse les modifications vers GitHub

**git checkout** : Littéralement “check out” (retirer, rapatrier) une branche.

**git -b <NomBranche>** : Crée une nouvelle branche dans votre dépôt.

**git log**: affiche un journal des modifications au dépôt

**git clone <URL PROVENANT DE GITHUB>** : clone un projet de votre système à partir de github

**git branch <nom\_branche>** : crée une copie de la branche master appelée

**git checkout <branch\_name>** : bascule vers nom\_branche comme branche de travail

**git checkout master** : bascule la branche vers master

**git branch -a** : affiche les branches existantes pour le dépôt particulier

**Liens et Ressources :**

Le [guide d’Atlassian](https://www.atlassian.com/git/tutorial/git-basics) a de très bons tutoriels en langage clair pour mieux comprendre et compléter cette anti-sèche de Git.

Pour les traductions des commandes, se référer à l’index de l’ouvrage de référence sur Git traduit en français : [git-scm/book](http://git-scm.com/book/fr)

**Glossaire ligne de commande**

**ls**: La liste des fichiers et dossiers installés dans votre dossier utilisateur apparaît.

**ls -a** : La liste de TOUS les éléments du dossier utilisateur, y compris les éléments invisibles, apparaît.

**ls -a Documents** : La liste de TOUS les éléments du dossier “Documents” apparaît (vous pouvez substituer “Documents” par un autre nom de dossier.

**ls -l** : Donne plus de détails que la commande ls : Apparaissent les permissions, le propriétaire, le groupe etc…

**man** : Cette commande vous indique à quoi correspond telle ou telle commande du terminal. Par exemple, tapez man ls : Cela vous indiquera à quoi correspond la commande ls

**pwd** : (Print Working Directory) Cette commande affiche le chemin absolu vers le répertoire dans lequel vous vous trouvez. Ceci vous permettra de vous localiser à tout moment dans l’arborescence OS X.

**cd** : (Change Directory) Pour changer de répertoire. Par exemple pour se rendre dans le répertoire de Marc, on tapera cd /Users/Marc

**cp** : Sert à copier un fichier : Par exemple pour copier le fichier User/Marc/documents/texte.doc dans le répertoire “important”, il faudra taper la commande : cp /Users/Marc/Documents/texte.doc Users/Marc/Documents/Important/texte.doc

**mv** : Sert à déplacer un fichier d’un répertoire vers un autre. (Même procédure que pour copier un fichier)

**rm** : Sert à supprimer définitivement un fichier. Par exemple : rm texte.doc

**rmdir** : Sert à supprimer définitivement un dossier vide. Par exemple : rmdir tartampion

**top** : Permet de voir les process en cours. Pour quitter taper “q”.

**df -h** : Affiche la liste des volumes montés sur votre Mac et les caractéristiques de ceux-ci (très pratique).

**chown** : Pour changer le propriétaire d’un fichier.

**mkdir** : Pour créer un répertoire.