

# GitHub Pour les Nuls : Pas de panique ; Lancez-vous ! (Première et seconde partie)

---

Dimanche 15 décembre 2013 par Christophe Ducamp

Traduction d'un article original de [Lauren Orsini](#) publié le 30 septembre 2013 pour ReadWriteWeb. Seul le [lien original fait référence](#).

La traduction reste à raffiner avec la pratique de cet outil. [Seconde partie en cours d'étude](#) pour me lancer sous peu dans les premiers *commits* à la ligne de commande. Mise en forme prévue pour le plan de route [indieweb](#) 2014. Merci. - [xtof\\_fr](#)

<https://stackoverflow.com/questions/783811/getting-git-to-work-with-a-proxy-server>

! Pour combattre les effets du proxy au boulot :

```
$ git config --list --global --show-origin
```

Cette commande retourne l'emplacement du fichier, et les lignes qui le compose ; typiquement, on a un fichier `c:\users\<vote nom de user>\.gitconfig`

Lui ajouter

```
[url "https://"]
  insteadOf = git://
```

## Table des matières

1.1	<i>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.</i>	3
1.2	Chercher des Réponses GitHub	3
1.3	C'est Quoi Git ?	4
1.4	Les Mots que les Personnes Utilisent quand Elles Parlent de Git	5
1.5	Commandes Spécifiques Git	6
1.6	Paramétrer GitHub ET Git Pour La Première Fois	7
1.7	Créez Votre Dépôt En Ligne	8
1.8	Créer Votre Dépôt Local	9
2	GitHub pour les Débutants : Consignez, Poussez et Foncez !	10
2.1	<i>Maintenant que nous connaissons les concepts Git, il est temps de jouer. Voici venue la deuxième partie de notre série.</i>	10
2.2	Connecter Votre Dépôt Local Vers Votre Dépôt GitHub	11
2.3	Tous ensemble Maintenant !	13
2.4	Les Ressources Git	14

## **1.1 *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](#) et [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 ?

## **1.2 Chercher des Réponses GitHub**

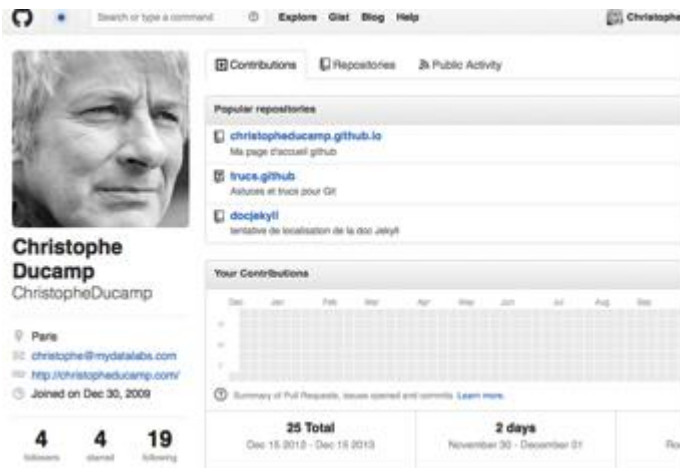
Aussi gênant que cela puisse paraître, j'ai écrit ce tutoriel parce que je me sentais vraiment perdue dans tous les articles de type "GitHub pour Débutants". Probablement parce qu'à la différence de la plupart des utilisateurs de Github, je manque de bagage solide en programmation. Et je ne m'y retrouvais 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](#)

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 pouvez déjà avoir plus d'une dizaine de comptes sociaux... voici pourquoi vous devriez être sur Github : il dispose des meilleures Conditions Générales d'Utilisation. Si vous regardez la section F des [conditions générales](#), 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](#)

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.

### 1.3 C'est Quoi Git ?

Remercions le célèbre développeur de logiciel [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](#) 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.

```
christopheducamp.github.io — bash — 80x10
Last login: Sun Dec 15 14:57:46 on ttys000
MacBook-Pro-de-Christophe:~ christopheducamp$ cd christopheducamp.github.io
MacBook-Pro-de-Christophe:christopheducamp.github.io christopheducamp$
```

GitHub facilite l'utilisation de Git sur deux points. Premièrement, si vous [téléchargez le logiciel GitHub](#) 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 emmène vos projets sur le web avec des contrôles de versions, et leur attache des fonctionnalités de réseaux sociaux.

Vous pouvez naviguer sur les projets d'autres utilisateurs de Github, et même y télécharger des copies pour vous afin de les modifier, apprendre ou les enrichir. D'autres utilisateurs peuvent faire de même avec vos projets publics, repérer vos erreurs et vous suggérer des corrections. De toute façon, aucune donnée ne se perd parce que Git sauvegarde 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 pourquoi. Dans ce tutoriel, nous apprendrons comment utiliser Git à la ligne de commande.

## 1.4 Les Mots que les Personnes Utilisent quand Elles Parlent de Git

Dans ce tutoriel, il y a quelques mots que j'utiliserai sans arrêt, 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 ferons ça dans la section suivante). Dans les deux cas, au lieu d'utiliser une souris, vous saisissez des commandes basées sur le texte, connues comme des invites de commande.

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

**Contrôle de Version** : Basiquement, 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 sinon vous sauvegardez plusieurs versions. Avec Git, vous n'êtes plus obligé de faire ça. Il conserve des "instantanés" de chaque point dans l'historique d'un projet, par conséquent 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 checkpoint que vous pouvez ensuite évaluer de nouveau ou restaurer votre projet à tout état précédent.

**Branche** : Comment plusieurs personnes travaillent sur un projet en même temps sans que Git ne s'embrouille ? Généralement, elles se "débranchent" du projet principal avec leurs propres versions pleines de modifications qu'elles ont chacune produites de leur côté. Après avoir fait ça, il est temps de "fusionner" cette branche là avec le "master", le répertoire principal du projet.

## 1.5 Commandes Spécifiques Git

Le fait que Git ait été conçu avec un gros projet comme Linux, il existe un paquet de commandes Git. Néanmoins, pour utiliser les basiques de Git, vous n'aurez besoin de connaître que 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 simplement un répertoire normal. 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 ? Saisissez-ça dans la ligne de commande pour ramener les 21 commandes les plus courues 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, quels sont les changements ayant besoin d'être gardés en mémoire, et sur quelle branche du repository vous êtes en train de travailler.

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

`git commit` : la commande la plus importante de Git. Après avoir fait toute sorte de modification, vous saisissez 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 voulez produire des modifications de votre côté ? Cette commande vous permet de construire une nouvelle branche, ou une timeline de commits, de modifications et d'ajouts de fichiers qui sont complètement les vôtres. Votre titre va après la commande. Si vous vouliez créer une nouvelle branche appelée “chats”, vous saisissez `git branch chats`.

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

`git merge` : Quand vous avez terminé de travailler sur une branche, vous pouvez fusionner vos modifications vers la branche master, qui est visible pour tous les collaborateurs. `git merge cats` prendrait toutes les modifications que vous avez faites sur 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 sur Github, vous “push”ez les modifications vers Github avec cette commande.

`git pull` : Si vous travaillez sur votre ordinateur local, et su 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.

## 1.6 Paramétrer GitHub ET Git Pour La Première Fois



Premièrement, vous devrez [vous enregistrer](#) 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.](#)



Désormais, il est temps de passer à la ligne de commande. Sur Windows, ça veut dire démarrer l'app 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"
```

Naturellement, vous devrez remplacer “Votre Nom Ici” avec 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 les projets futurs.

Puis, dites-lui votre e-mail et assurez-vous que c'est le même email que vous avez utilisé quand vous vous êtes enregistré pour 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 démarrer Git sur votre ordinateur. Cependant, du fait que vous venez de créer un compte Github.com, il est probable que vous ne souhaitiez pas gérer simplement votre projet localement, mais aussi en ligne. Si vous voulez, vous pouvez aussi paramétrer Git de façon à ce qu'il ne vous demande pas de vous connecter à votre compte Github à chaque fois que vous voulez lui parler. Pour

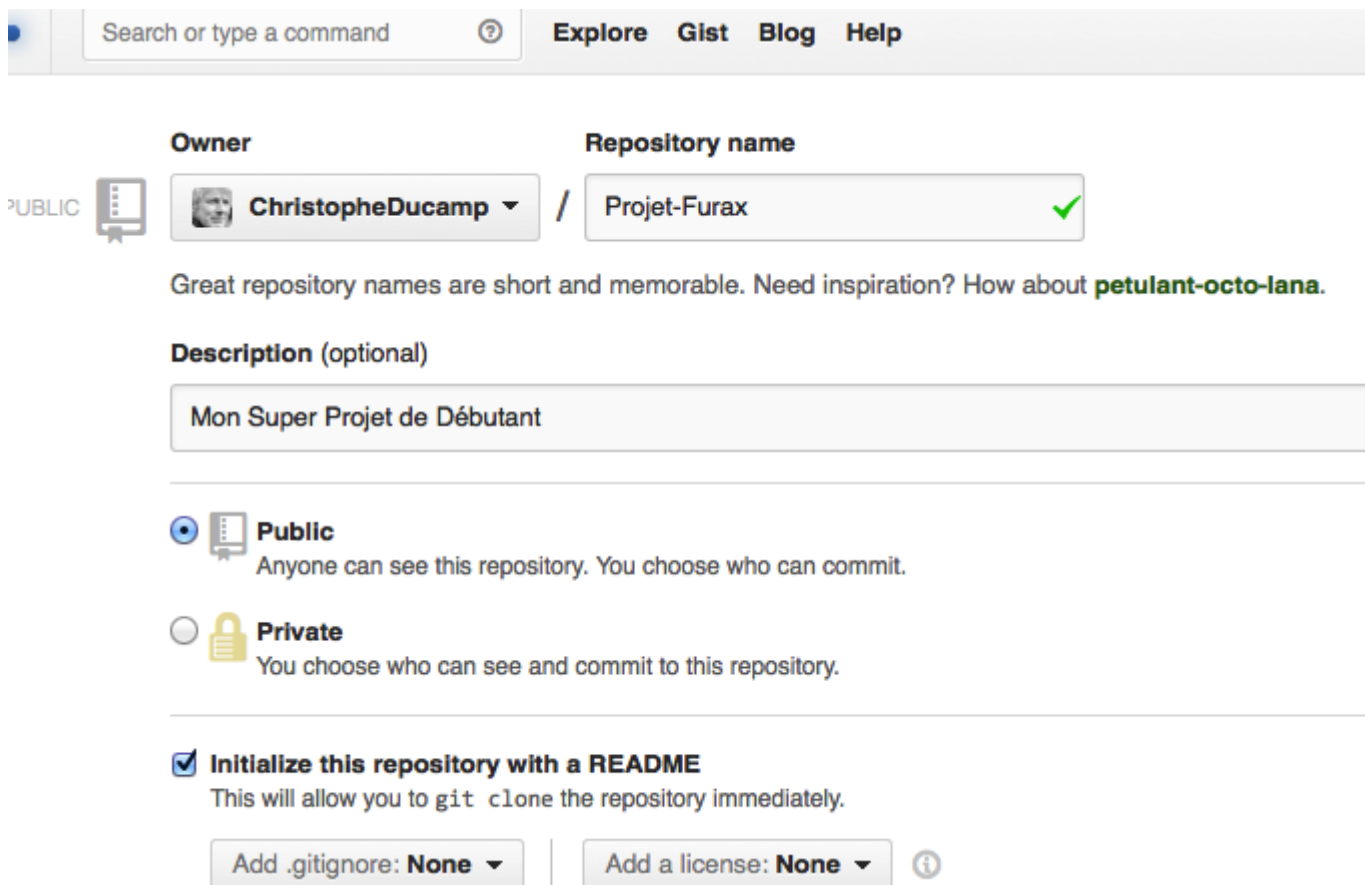
les besoins de ce tutoriel, ce n'est pas un grand problème parce que nous ne lui parlerons qu'une fois. Le tutoriel complet pour faire ça est [situé sur Github](#).

```
christopheducamp — bash — 165x9
Last login: Sun Dec 15 14:59:34 on ttys000
MacBook-Pro-de-Christophe:~ christopheducamp$ git config --global user.name "Christophe Ducamp"
MacBook-Pro-de-Christophe:~ christopheducamp$ git config --global user.email "christophe@mydatalabs.com"
MacBook-Pro-de-Christophe:~ christopheducamp$
```

## 1.7 Créez Votre Dépôt En Ligne


Maintenant que vous avez tout réglé, il est temps de créer un endroit pour placer votre projet à faire vivre. Tant Git que Github font référence à cela en tant que repository, ou “repo” pour le raccourci, un répertoire digital ou un espace de stockage où vous pouvez accéder à votre projet, ses fichiers, et toutes les versions de ses fichiers que Git a sauvegardés.

Revenons 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, n'ayez pas peur de montrer votre tentative d'apprendre Github !



Search or type a command ? Explore Gist Blog Help

**Owner** **Repository name**

PUBLIC  **ChristopheDucamp** / **Projet-Furax** ✓

Great repository names are short and memorable. Need inspiration? How about **petulant-octo-lana**.

**Description (optional)**

Mon Super Projet de Débutant

☒ **Public**  
Anyone can see this repository. You choose who can commit.

☐ **Private**  
You choose who can see and commit to this repository.

☒ **Initialize this repository with a README**  
This will allow you to `git clone` the repository immediately.

Add .gitignore: **None** | Add a license: **None** ⓘ

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`.

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



## 1.8 Créer Votre Dépôt Local

Ainsi vous venons juste de créer un espace pour votre projet en ligne, mais ce n'est pas l'endroit où nous travaillerons dessus. Le plus gros de votre travail sera produit sur votre ordinateur. Par conséquent, nous devons en fait refléter ce repository que nous venons juste de produire sous un répertoire local.

C'est l'endroit où nous produisons quelques saisies de ligne de commande faisant 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 `~/` vous assure que nous construisons le dépôt tout en haut de la structure du fichier 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 répertoire Users.

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

Puis, saisissez :

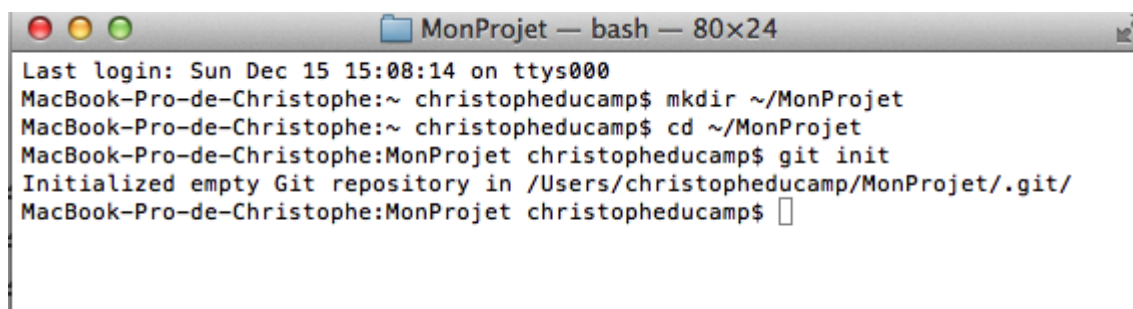
```
cd ~/MonProjet
```

`cd` signifie change directory, et c'est aussi une commande de navigation. Nous venons de produire un répertoire, et nous voulons maintenant migrer vers ce répertoire et aller dedans. Une fois que nous avons tapé cette 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.



```
MonProjet — bash — 80x24
Last login: Sun Dec 15 15:08:14 on ttys000
MacBook-Pro-de-Christophe:~ christopheducamp$ mkdir ~/MonProjet
MacBook-Pro-de-Christophe:~ christopheducamp$ cd ~/MonProjet
MacBook-Pro-de-Christophe:MonProjet christopheducamp$ git init
Initialized empty Git repository in /Users/christopheducamp/MonProjet/.git/
MacBook-Pro-de-Christophe:MonProjet christopheducamp$ █
```

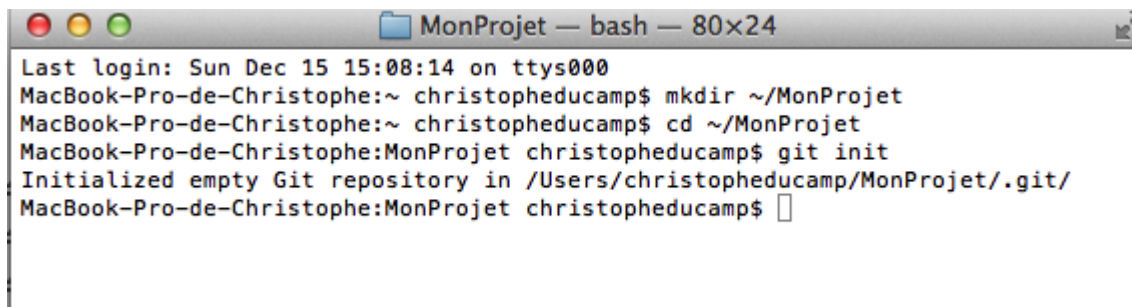
Néanmoins, votre ordinateur réalise maintenant que ce répertoire est *git-ready*, et vous pouvez commencer à entrer des commandes Git. Maintenant, vous disposez à la fois d'un dépôt local et en ligne pour votre projet. Dans la [seconde partie](#), vous apprendrez comment faire votre premier commit vers des dépôts locaux et sur Github. Et vous en saurez plus sur quelques ressources géniales de Github.

## 2 GitHub pour les Débutants : Consignez, Poussez et Foncez !

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

Dans la [1ère</sup> partie de ce tutoriel GitHub en deux parties](/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 dépôt local pour le code.

Ces premières étapes étant accomplies, ajoutons désormais la première partie de notre projet en **produisant notre premier “commit” sur GitHub**. À la fin de la première partie, nous avons créé un dépôt local appelé MonProjet, qui, visualisé à la ligne de commande, ressemble à cette impression-écran :



```
MonProjet — bash — 80x24
Last login: Sun Dec 15 15:08:14 on ttys000
MacBook-Pro-de-Christophe:~ christopheducamp$ mkdir ~/MonProjet
MacBook-Pro-de-Christophe:~ christopheducamp$ cd ~/MonProjet
MacBook-Pro-de-Christophe:MonProjet christopheducamp$ git init
Initialized empty Git repository in /Users/christopheducamp/MonProjet/.git/
MacBook-Pro-de-Christophe:MonProjet christopheducamp$
```

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

```
$ touch Lisez-moi.txt
```

Une fois de plus, *ceci n'est pas* une commande Git. C'est simplement un autre standard de navigation (une “invite de commande”). `touch` signifie en fait “créer”. Tout ce que vous écrivez après, 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 `Lisez-moi.txt` se niche désormais à l'intérieur. Vous auriez pu varier le plaisir avec quelque chose comme “Lisez-moi.doc” ou “Wiki.gif,”.

Désormais, vous pouvez voir clairement votre nouveau fichier `Lisez-moi`. Mais Git le peut-il aussi ? Regardons ça. Tapez :

```
$ git status
```

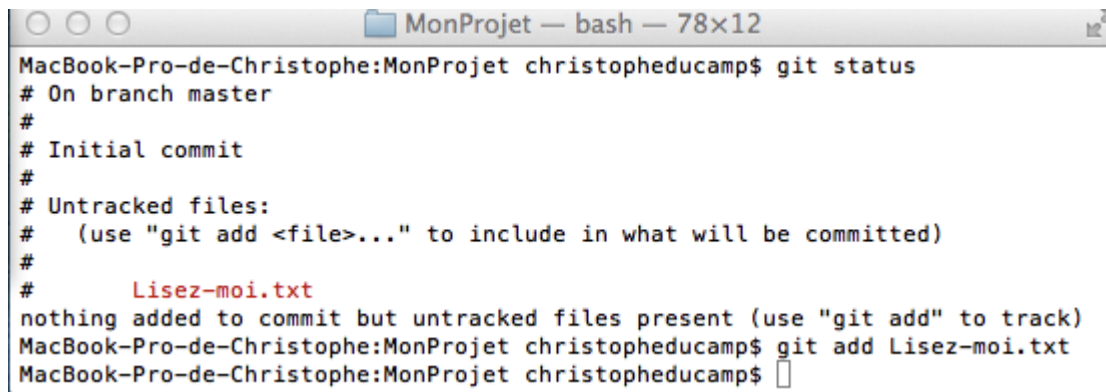
La ligne de commande, généralement passive jusqu'à ce stade, vous renvoie quelques lignes de texte similaires à ce qui suit :

```
# On branch master
#
# Untracked files:
#
#   (use "git add ..." to include in what will be committed)
#
#       Lisez-moi.txt
```

Que se passe-t-il ?

Tout d’abord, vous êtes sur la branche `master` de votre projet, ce qui a du sens puisque nous ne l’avons pas “débranchée” du projet. Il n’y a aucune raison à cela, puisque nous travaillons seul. Deuxièmement, `Lisez-moi.txt` est listé comme un fichier “untracked”, ce qui signifie que Git l’ignore à cette heure. Pour faire en sorte que Git remarque que le fichier est là, saisissez :

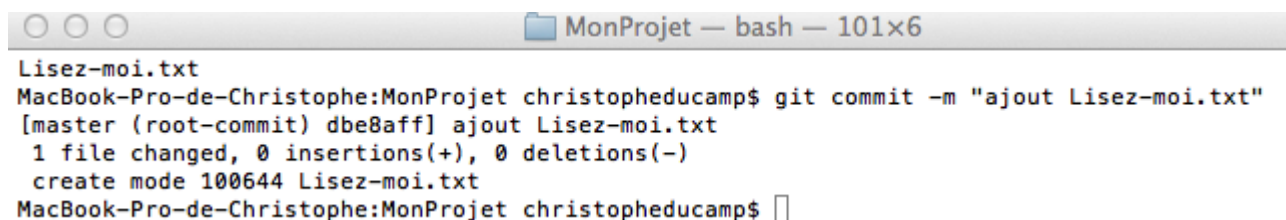
```
$ git add Lisez-moi.txt
```



```
MacBook-Pro-de-Christophe:MonProjet christopheducamp$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       Lisez-moi.txt
nothing added to commit but untracked files present (use "git add" to track)
MacBook-Pro-de-Christophe:MonProjet christopheducamp$ git add Lisez-moi.txt
MacBook-Pro-de-Christophe:MonProjet christopheducamp$
```

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"
```



```
Lisez-moi.txt
MacBook-Pro-de-Christophe:MonProjet christopheducamp$ git commit -m "ajout Lisez-moi.txt"
[master (root-commit) dbe8aff] ajout Lisez-moi.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 Lisez-moi.txt
MacBook-Pro-de-Christophe:MonProjet christopheducamp$
```

Le marqueur `m` comme indiqué dans le répertoire des définitions en [1<sup>ère</sup> partie](#), indique simplement que le texte qui suit devrait être lu comme un message. Remarquez que le message `commit` est écrit au temps 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 produit précédemment, parce que vous pouvez toujours revenir à la version précédente. **Vous écrivez ce que fait un commit.**

Maintenant que nous avons produit localement un petit travail, il est temps d’envoyer (de “push”er) notre premier “commit” sur GitHub.

“Attendez, nous n’avons jamais connecté mon dépôt en ligne vers mon dépôt local,” pourriez-vous penser. Et vous avez raison. En fait, votre dépôt local et votre dépôt en ligne ne se connectent que par épisodes courts et intermittents, quand vous confirmez les ajouts et modifications au projet. Avançons pour produire maintenant notre première vraie connexion.

## 2.2 Connecter Votre Dépôt Local Vers Votre Dépôt GitHub

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

Ce paramétrage facilite aussi le fait d’avoir plusieurs collaborateurs oeuvrant sur le même projet. Chacun d’entre 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.

Premièrement, 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.

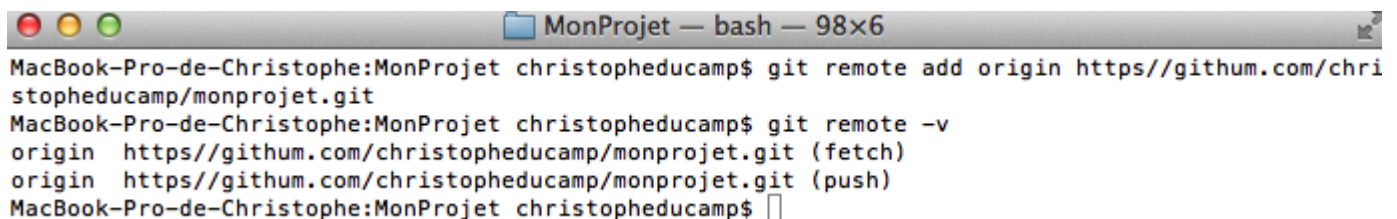
Supposons que nous ayons un dépôt GitHub appelé "MonProjet" situé sur `https://github.com/nomutilisateur/MonProjet.git`. Bien sûr, `nomutilisateur` devrait être remplacé par votre véritable nom d'utilisateur Github, et `MonProjet` devrait être remplacé par le véritable titre que vous avez donné à votre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

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

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
```



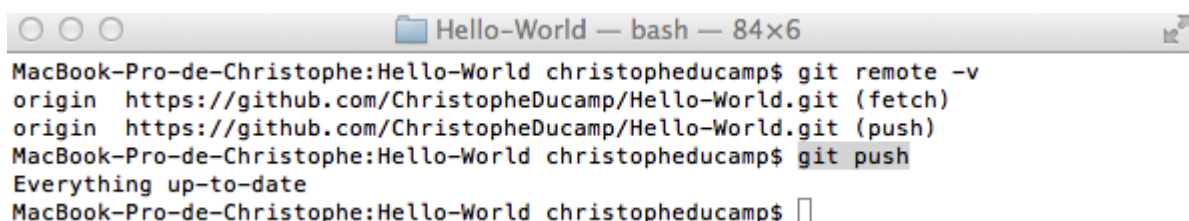
```
MacBook-Pro-de-Christophe:MonProjet christopheducamp$ git remote add origin https://github.com/christopheducamp/monprojet.git
MacBook-Pro-de-Christophe:MonProjet christopheducamp$ git remote -v
origin https://github.com/christopheducamp/monprojet.git (fetch)
origin https://github.com/christopheducamp/monprojet.git (push)
MacBook-Pro-de-Christophe:MonProjet christopheducamp$
```

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."



```
MacBook-Pro-de-Christophe>Hello-World christopheducamp$ git remote -v
origin https://github.com/ChristopheDucamp/Hello-World.git (fetch)
origin https://github.com/ChristopheDucamp/Hello-World.git (push)
MacBook-Pro-de-Christophe>Hello-World christopheducamp$ git push
Everything up-to-date
MacBook-Pro-de-Christophe>Hello-World christopheducamp$
```

Git aurait pu ici me renvoyer ici un paquet d'avertissements parce que j'ai simplement produit la commande simple. Si j'avais voulu être plus spécifique, j'aurais pu saisir `push origin master`, pour spécifier que je voulais dire la branche `master` de mon dépôt. Je n'ai pas fait ça parce que je n'avais qu'une branche à ce stade.

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 `Lisez-moi.txt` car nous l'avons construit précédemment à l'intérieur de votre dépôt local.

## 2.3 Tous ensemble Maintenant !

Bravo, vous êtes officiellement un utilisateur Git ! Vous pouvez créer des dépôts et *commit* 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>)

Néanmoins, vous pouvez avoir cet étrange sentiment que vous n'êtes pas encore expert. Bien sûr, vous êtes parvenu(e) 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 avez construit un nouveau site web pour le Magasin de Glaces de Jean avec quelques-uns de vos collègues.

Vous étiez un peu nerveux quand votre boss vous a demandé de participer au projet de redesign de la page web du Magasin de Glaces de Jean. Après tout, vous n'êtes pas programmeur ; vous êtes designer graphique. Mais votre boss 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 sauvegardées dans un répertoire sur votre ordinateur, appelé "icecream", pour éviter de vous emmêler.

Ouvrez la Ligne de Commande et changez de répertoire jusqu'à ce vous soyez dans le répertoire `icecream`, là où sont stockés vos designs.

```
$ 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 (username: 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.

## 2.4 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](#), voici quelques ressources que j'ai trouvées utiles tout en apprenant personnellement à utiliser Git et Github durant l'été :

- [Pro Git](#). 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](#). 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](#). 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](#) en quatre parties.
- [Git Reference](#). Vous avez les basiques mais vous oubliez toujours les commandes ? Ce site pratique est génial comme glossaire de référence.
- [Git - le petit guide](#). 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.