**--- GIT ---**

**C’est quoi Git ?**

**Git est un logiciel de contrôle de version (version control) qui permet d’avoir un historique les modifications apportées a son code ou a celui des autres. Ce projet open-source gratuit est un standard de l’industrie que tout développeur se doit de connaître sur le bout des doigts !**

**Sommaire**

[**Commandes GIT de base**](https://www.hostinger.fr/tutoriels/commandes-git#Commandes_GIT_de_base)

* + [**Git config**](https://www.hostinger.fr/tutoriels/commandes-git#Git_config)
  + [**Git init**](https://www.hostinger.fr/tutoriels/commandes-git#Git_init)
  + [**Git add**](https://www.hostinger.fr/tutoriels/commandes-git#Git_add)
  + [**Git clone**](https://www.hostinger.fr/tutoriels/commandes-git#Git_clone)
  + [**Git commit**](https://www.hostinger.fr/tutoriels/commandes-git#Git_commit)
  + [**Git status**](https://www.hostinger.fr/tutoriels/commandes-git#Git_status)
  + [**Git push**](https://www.hostinger.fr/tutoriels/commandes-git#Git_push)
  + [**Git checkout**](https://www.hostinger.fr/tutoriels/commandes-git#Git_checkout)
  + [**Git remote**](https://www.hostinger.fr/tutoriels/commandes-git#Git_remote)
  + [**Branche git**](https://www.hostinger.fr/tutoriels/commandes-git#Branche_git)
  + [**Git pull**](https://www.hostinger.fr/tutoriels/commandes-git#Git_pull)
  + [**Git merge**](https://www.hostinger.fr/tutoriels/commandes-git#Git_merge)
  + [**Git diff**](https://www.hostinger.fr/tutoriels/commandes-git#Git_diff)
  + [**Git tag**](https://www.hostinger.fr/tutoriels/commandes-git#Git_tag)
  + [**Git log**](https://www.hostinger.fr/tutoriels/commandes-git#Git_log)
  + [**Git reset**](https://www.hostinger.fr/tutoriels/commandes-git#Git_reset)
  + [**Git rm**](https://www.hostinger.fr/tutoriels/commandes-git#Git_rm)
  + [**Git stash**](https://www.hostinger.fr/tutoriels/commandes-git#Git_stash)
  + [**Git show**](https://www.hostinger.fr/tutoriels/commandes-git#Git_show)
  + [**Git fetch**](https://www.hostinger.fr/tutoriels/commandes-git#Git_fetch)
  + [**Git ls-tree**](https://www.hostinger.fr/tutoriels/commandes-git#Git_ls-tree)
  + [**Git cat-file**](https://www.hostinger.fr/tutoriels/commandes-git#Git_cat-file)
  + [**Git grep**](https://www.hostinger.fr/tutoriels/commandes-git#Git_grep)
  + [**Gitk**](https://www.hostinger.fr/tutoriels/commandes-git#Gitk)
  + [**Git instaweb**](https://www.hostinger.fr/tutoriels/commandes-git#Git_instaweb)
  + [**Git gc**](https://www.hostinger.fr/tutoriels/commandes-git#Git_gc)
  + [**git archive**](https://www.hostinger.fr/tutoriels/commandes-git#git_archive)
  + [**Git prune**](https://www.hostinger.fr/tutoriels/commandes-git#Git_prune)
  + [**Git fsck**](https://www.hostinger.fr/tutoriels/commandes-git#Git_fsck)
  + [**Git rebase**](https://www.hostinger.fr/tutoriels/commandes-git#Git_rebase)

**Git config**

* L’une des commandes git les plus utilisées est **git config.** On l’utilise pour configurer les préférences de l’utilisateur : son adresse email, l’algorithme utilisé pour diff, le nom d’utilisateur et le format de fichier etc. Par exemple, la commande suivante peut être utilisée pour définir le mail d’un utilisateur:

git config --global user.email sam@google.com

**Git init**

* Cette commande est utilisée pour créer un nouveau dépôt [**GIT**](https://www.hostinger.fr/tutoriels/comment-utiliser-une-branche-git) :

git init

**Git add**

* La **commande git add** peut être utilisée pour ajouter des fichiers à l’index. Par exemple, la commande suivante ajoutera un fichier nommé temp.txt dans le répertoire local de l’index:

git add temp.txt

**Git clone**

* La **commande git clone** est utilisée pour la vérification des dépôts. Si le dépôt se trouve sur un serveur distant, utilisez:

git clone alex@93.188.160.58:/chemin/vers/dépôt

* Inversement, si une copie de travail d’un dépôt local doit être créée, utilisez:

git clone /chemin/vers/dépôt

**Git commit**

* La **commande git commit** permet de **valider les modifications apportées** au HEAD. Notez que tout commit ne se fera pas dans le dépôt distant.

git commit –m “Description du commit”

**Git status**

* La **commande git status** affiche la liste des fichiers modifiés ainsi que les fichiers qui doivent encore être ajoutés ou validés. Usage:

git status

**Git push**

* **Git push** est une autre commandes [**GIT**](https://www.hostinger.fr/tutoriels/meilleurs-clients-git-gui/) de base. Un simple push envoie les modifications locales apportées à la branche principale associée :

git push origin master

**Git checkout**

* La **commande git checkout** peut être utilisée pour créer des branches ou pour basculer entre elles. Par exemple nous allons créer une branche:

command git checkout -b <nom-branche>

* Pour passer simplement d’une branche à une autre, utilisez:

git checkout <nom-branche>

**Git remote**

* Cette commande**remote** permet à un utilisateur de se connecter à un dépôt distant. La commande suivante répertorie les dépôts distants actuellement configurés:

git remote –v

* Cette commande permet à l’utilisateur de connecter le dépôt local à un serveur distant:

git remote add origin <93.188.160.58>

**Branche git**

* La **commande git branch** peut être utilisée pour répertorier, créer ou supprimer des branches. Pour répertorier toutes les branches présentes dans le dépôt, utilisez:

git branch

* Pour supprimer une branche:

git branch –d <nom-branche>

**Git pull**

* Pour fusionner toutes les modifications présentes sur le dépôt distant dans le répertoire de travail local, la commande pull est utilisée. Usage:

git pull

**Git merge**

* La **commande git merge** est utilisée pour fusionner une branche dans la branche active. Usage:

git merge <nom-branche>444444111111111

**Git diff**

* La **commande git diff** permet de lister les conflits. Pour visualiser les conflits d’un fichier, utilisez

git diff --base <nom-fichier>

* La commande suivante est utilisée pour afficher les conflits entre les branches à fusionner avant de les fusionner:

git diff <branche-source> <branche-cible>

* Pour simplement énumérer tous les conflits actuels, utilisez:

git diff

**Git tag**

* Le marquage est utilisé pour marquer des commits spécifiques avec des poignées simples. Un exemple peut être:

git tag 1.1.0 <insert-commitID-here>

**Git log**

* L’ **exécution de** cette commande génère le log d’une branche. Un exemple de sortie :
* commit 15f4b6c44b3c8344caasdac9e4be13246e21sadw
* Author: Alex Hunter <alexh@gmail.com>

Date: Mon Oct 1 12:56:29 2016 -0600

**Git reset**

* Pour réinitialiser l’index et le répertoire de travail à l’état du dernier commit, la **commande git reset** est utilisée :

git reset --hard HEAD

**Git rm**

* **Git rm** peut être utilisé pour supprimer des fichiers de l’index et du répertoire de travail. Usage:

git rm nomfichier.txt

**Git stash**

* L’une des moins connues, **git stash** aide à enregistrer les changements qui ne doivent pas être commit immédiatement. C’est un commit temporaire. Usage:

git stash

**Git show**

* Pour afficher des informations sur tout fichier git, utilisez la **commande git show** . Par exemple:

git show

**Git fetch**

* **Git fetch** permet à un utilisateur d’extraire tous les fichiers du dépôt distant qui ne sont pas actuellement dans le répertoire de travail local. Exemple d’utilisation:

git fetch origin

**Git ls-tree**

* Pour afficher un fichier arborescent avec le nom et le mode de chaque élément, et la valeur SHA-1 du blob, utilisez la **commande git ls-tree** . Par exemple:

git ls-tree HEAD

**Git cat-file**

* À l’aide de la valeur SHA-1, affichez le type d’un fichier à l’aide de la **commande git cat-file** . Par exemple:

git cat-file –p d670460b4b4aece5915caf5c68d12f560a9fe3e4

**Git grep**

* **Git grep** permet à un **utilisateur de rechercher** dans les arbres de contenu des expressions et / ou des mots. Par exemple, pour *rechercher www.hostinger.com* dans tous les fichiers, utilisez:

git grep "www.hostinger.com"

**Gitk**

* **Gitk** est l’interface graphique du dépôt local. Vous pouvez l’appeler en exécutant:

gitk

**Git instaweb**

* Avec la **commande git instaweb** , un serveur Web peut être exécuté par interface avec le dépôt local. Qui redirige directement vers un serveur web. Par exemple:

git instaweb –httpd=webrick

**Git gc**

* Pour optimiser le dépôt en supprimant les fichiers inutiles et les optimiser, utilisez:

git gc

**git archive**

* La **commande git archive** permet à un utilisateur de créer un fichier zip ou tar contenant les composants d’un arbre du dépôt. Par exemple:

git archive --format=tar master

**Git prune**

* Via la **commande git prune** , les fichiers qui n’ont pas de pointeurs entrants seront supprimés. Usage:

git prune

**Git fsck**

* Pour effectuer une vérification d’intégrité du système de fichiers git, utilisez la commande **git fsck** . Tous les fichiers corrompus seront identifiés:

git fsck

**Git rebase**

* La **commande git rebase** est utilisée pour la réapplication des commits sur une autre branche. Par exemple:

git rebase master