CONTENU

// Cheatsheet //

How To Use Git: A Reference Guide

Published on October 10, 2018 · Updated on September 2, 2021

Git Open Source Development

By Lisa Tagliaferri

English

Git Cheat Sheet

Introduction

Teams of developers and open-source software maintainers typically manage their projects through Git, a distributed version control system that supports collaboration.

This cheat sheet style guide provides a quick reference to commands that are useful for working and collaborating in a Git repository. To install and configure Git, be sure to read "How To Contribute to Open Source: Getting Started with Git."

How to Use This Guide:

- This guide is in cheat sheet format with self-contained command-line snippets.
- Jump to any section that is relevant to the task you are trying to complete.
- When you see in this guide's commands, keep in mind that this text should refer to the commits and files in *your own* repository. `highlighted text`

Set Up and Initialization

Check your Git version with the following command, which will also confirm that Git is installed:

```
$ git --version
```

Copy

Git allows you to configure a number of settings that will apply to all the repositories on your local machine. For instance, configure a username that Git will use to credit you with any changes you make to a local repository:

```
$ git config --global user.name " firstname lastname "
```

Copy

Configure an email address to be associated with each history marker:

```
$ git config --global user.email " valid-email "
```

Copy

Configure your preferred text editor as well:

```
$ git config --global core.editor " nano "
```

Copy

You can initialize your current working directory as a Git repository with : `init`

```
$ git init
```

Copy

To copy an existing Git repository hosted remotely, you'll use with the repo's URL or server location (in the latter case you will use): `git clone ssh`

```
$ git clone  https://www.github.com/username/repo-name
```

Copy

Show your current Git directory's remote repository:

```
$ git remote
```

Copier

Pour une sortie plus détaillée, utilisez l'indicateur : `-v`

```
$ git remote -v
```

Copier

Ajoutez le Git en amont, qui peut être une URL ou peut être hébergé sur un serveur (dans ce dernier cas, connectez-vous avec) : `ssh`

```
$ git remote add upstream  https://www.github.com/username/repo-name
```

Copier

Mise en scène

Lorsque vous avez modifié un fichier et que vous l'avez marqué pour qu'il aille dans votre prochain commit, il est considéré comme un fichier intermédiaire.

Vérifiez l'état de votre référentiel Git, y compris les fichiers ajoutés qui ne sont pas transférés et les fichiers qui sont transférés :

```
$ git status
```

Copier

Pour préparer des fichiers modifiés, utilisez la commande, que vous pouvez exécuter plusieurs fois avant un commit. Si vous apportez des modifications ultérieures que vous souhaitez inclure dans la validation suivante, vous devez l'exécuter à nouveau. `add add`

Vous pouvez spécifier le fichier spécifique avec : `add`

```
$ git add  my_script.py
```

Copier

Avec vous pouvez ajouter tous les fichiers dans le répertoire courant, y compris les fichiers qui commencent par un : `. .`

```
$ git add .
```

Copier

Si vous souhaitez ajouter tous les fichiers d'un répertoire courant ainsi que des fichiers dans des sous-répertoires, vous pouvez utiliser l'indicateur ou : `-all -A`

```
$ git add -A
```
Copier

Vous pouvez supprimer un fichier de la préproduction tout en conservant les modifications dans votre répertoire de travail avec : `reset`

```
$ git reset  my_script.py
```
Copier

Commettre

Une fois que vous avez préparé vos mises à jour, vous êtes prêt à les valider, ce qui enregistrera les modifications que vous avez apportées au référentiel.

Pour valider des fichiers intermédiaires, vous allez exécuter la commande avec votre message de validation significatif afin de pouvoir suivre les validations : `commit`

```
$ git commit -m "Commit message"
```
Copier

Vous pouvez condenser le transfert de tous les fichiers suivis en les validant en une seule étape :

```
$ git commit -am "Commit message"
```
Copier

Si vous avez besoin de modifier votre message de validation, vous pouvez le faire avec l'indicateur : `--amend`

```
$ git commit --amend -m "New commit message"
```
Copier

Branches

Une branche dans Git est un pointeur mobile vers l'un des commits du dépôt, elle vous permet d'isoler le travail et de gérer le développement et les intégrations de fonctionnalités. Pour en savoir plus sur les branches, consultez la documentation Git.

Répertoriez toutes les branches actuelles à l'aide de la commande. Un astérisque () apparaîtra à côté de votre branche actuellement active : `branch *`

```
$ git branch
```
Copier

Créez une nouvelle branche. Vous resterez sur votre branche active jusqu'à ce que vous passiez à la nouvelle :

```
$ git branch  new-branch
```
Copier

Basculez vers n'importe quelle branche existante et extrayez celle-ci dans votre répertoire de travail actuel :

```
$ git checkout  another-branch
```
Copier

Vous pouvez consolider la création et l'extraction d'une nouvelle branche à l'aide de l'indicateur : `-b`

```
$ git checkout -b  new-branch
```
Copier

Renommez le nom de votre branche :

```
$ git branch -m  current-branch-name  new-branch-name
```
Copier

Fusionnez l'historique de la branche spécifiée dans celui dans lequel vous travaillez actuellement :

```
$ git merge  branch-name
```
Copier

Abandonnez la fusion, en cas de conflit :

```
$ git merge --abort
```
Copier

Vous pouvez également sélectionner un commit particulier avec lequel fusionner avec la chaîne qui fait référence au commit spécifique : `cherry-pick`

```
$ git cherry-pick  f7649d0
```
Copier

Lorsque vous avez fusionné une branche et que vous n'en avez plus besoin, vous pouvez la supprimer :

```
$ git branch -d  branch-name
```
Copier

Si vous n'avez pas fusionné une branche vers main, mais que vous êtes sûr de vouloir la supprimer, vous pouvez **forcer** la suppression d'une branche :

```
$ git branch -D  branch-name
```

Copier

Collaborer et mettre à jour

Pour télécharger les modifications à partir d'un autre référentiel, tel que le dépôt distant en amont, vous utiliserez : `fetch`

```
$ git fetch  upstream
```

Copier

Fusionnez les commits récupérés. Notez que certains dépôts peuvent utiliser à la place de : `master main`

```
$ git merge upstream/main
```

Copier

Envoyez ou transmettez vos validations de branche locale à la branche du référentiel distant :

```
$ git push origin main
```

Copier

Récupérez et fusionnez tous les commits à partir de la branche distante de suivi :

```
$ git pull
```
Copier

Inspection

Affichez l'historique des validations de la branche actuellement active :

```
$ git log
```
Copier

Affichez les validations qui ont modifié un fichier particulier. Cela suit le fichier quel que soit le renommage du fichier :

```
$ git log --follow  my_script.py
```
Copier

Affichez les commits qui se trouvent sur une branche et pas sur l'autre. Cela affichera les commits sur qui ne sont pas sur : `a-branch  b-branch`

```
$ git log  a-branch .. b-branch
```
Copier

Consultez les journaux de référence () pour voir quand les pointes des branches et autres références ont été mises à jour pour la dernière fois dans le référentiel : `reflog`

```
$ git reflog
```
Copier

Affichez n'importe quel objet dans Git via sa chaîne de validation ou son hachage dans un format plus lisible par l'homme :

```
$ git show  de754f5
```

Copier

Afficher les modifications

La commande affiche les changements entre les commits, les branches, etc. Vous pouvez en savoir plus à ce sujet dans la documentation Git. `git diff`

Comparez les fichiers modifiés qui se trouvent dans la zone de transit :

```
$ git diff --staged
```

Copier

Afficher le diff de ce qui est dedans mais n'est pas dedans : `a-branch` `b-branch`

```
$ git diff  a-branch .. b-branch
```

Copier

Suivez les modifications de chemin d'accès en supprimant un fichier de votre projet et en préparant cette suppression pour validation :

```
$ git rm  file
```
Copier

Vous pouvez également modifier un chemin d'accès à un fichier existant, puis effectuer le déplacement :

```
$ git mv  existing-path   new-path
```
Copier

Vérifiez le journal de validation pour voir si des chemins ont été déplacés :

```
$ git log --stat -M
```
Copier

Cachette

Parfois, vous constaterez que vous avez apporté des modifications à certains codes, mais avant de terminer, vous devez commencer à travailler sur autre chose. Vous n'êtes pas tout à fait prêt à valider les modifications que vous avez apportées jusqu'à présent, mais vous ne voulez pas perdre votre travail. La commande vous permettra d'enregistrer vos modifications locales et de revenir au répertoire de travail qui correspond au commit le plus récent. `git stash HEAD`

Rangez votre travail actuel :

```
$ git stash
```
Copier

Voyez ce que vous avez actuellement en réserve :

```
$ git stash list
```
Copier

Vos caches seront nommées , , et ainsi de suite. `stash@{0}` `stash@{1}`

Afficher des informations sur une réserve particulière :

```
$ git stash show stash@{ 0 }
```
Copier

Pour faire sortir les fichiers d'une réserve actuelle de la cachette tout en conservant la réserve, utilisez : `apply`

```
$ git stash apply stash@{ 0 }
```
Copier

Si vous souhaitez sortir des fichiers d'une cachette et que vous n'en avez plus besoin, utilisez : `pop`

```
$ git stash pop stash@{ 0 }
```
Copier

Si vous n'avez plus besoin des fichiers enregistrés dans une réserve particulière, vous pouvez : `drop`

```
$ git stash drop stash@{ 0 }
```
Copier

Si vous avez plusieurs cachettes enregistrées et que vous n'avez plus besoin d'en utiliser, vous pouvez utiliser pour les supprimer : `clear`

```
$ git stash clear
```
Copier

Ignorer les fichiers

Si vous souhaitez conserver des fichiers dans votre répertoire Git local, mais que vous ne souhaitez pas les valider dans le projet, vous pouvez ajouter ces fichiers à votre fichier afin qu'ils ne provoquent pas de conflits. `.gitignore`

Utilisez un éditeur de texte tel que nano pour ajouter des fichiers au fichier : `.gitignore`

```
$ nano .gitignore
```
Copier

Pour voir des exemples de fichiers, vous pouvez consulter le site GitHub `.gitignore` Référentiel de modèle.

Rebasage

Un rebase nous permet de déplacer les branches en changeant le commit sur lequel elles sont basées. Avec le rebasage, vous pouvez écraser ou reformuler les commits.

Vous pouvez démarrer un rebase en appelant le nombre de commits que vous avez effectués et que vous souhaitez rebaser (dans le cas ci-dessous) : `5`

```
$ git rebase -i HEAD~ 5
```
Copier

Vous pouvez également effectuer un rebasage en fonction d'une chaîne de validation ou d'un hachage particulier :

```
$ git rebase -i  074a4e5
```

Copier

Une fois que vous avez écrasé ou reformulé les commits, vous pouvez terminer le rebasage de votre branche en plus de la dernière version du code amont du projet. Notez que certains dépôts peuvent utiliser à la place de : `master main`

```
$ git rebase  upstream/main
```

Copier

Pour en savoir plus sur le rebasage et la mise à jour, vous pouvez lire Comment rebaser et mettre à jour une demande de tirage, qui s'applique également à tout type de validation.

Restauration et réinitialisation

Vous pouvez annuler les modifications que vous avez apportées à un commit donné à l'aide de . Pour que cela soit possible, votre arbre de travail devra être propre : `revert`

```
$ git revert  1fc6665
```

Copier

Parfois, y compris après un rebasage, vous devez réinitialiser votre arbre de travail. Vous pouvez réinitialiser un commit particulier et **supprimer toutes les modifications** à l'aide de la commande suivante :

```
$ git reset --hard  1fc6665
```
Copier

Pour forcer le push de votre dernier commit non conflictuel connu vers le dépôt d'origine, vous devez utiliser : `--force`

> **Avertissement** : Forcer à pousser vers la branche principale (parfois) est souvent mal vu, à moins qu'il n'y ait une raison vraiment importante de le faire. Utilisez-le avec parcimonie lorsque vous travaillez sur vos propres référentiels, et efforcez-vous d'éviter cela lorsque vous collaborez. `master`

```
$ git push --force origin main
```
Copier

Pour supprimer les fichiers locaux non suivis et les sous-répertoires du répertoire Git pour une branche de travail propre, vous pouvez utiliser : `git clean`

```
$ git clean -f -d
```
Copier

Si vous avez besoin de modifier votre dépôt local pour qu'il ressemble à la branche principale en amont actuelle (c'est-à-dire qu'il y a trop de conflits), vous pouvez effectuer une réinitialisation matérielle :

> **Remarque** : L'exécution de cette commande donnera à votre référentiel local l'apparence exacte de celui de l'amont. Tous les commits que vous avez faits mais qui n'ont pas été tirés vers l'amont **seront détruits**.

```
$ git reset --hard upstream/main
```

Copier

Conclusion

Ce guide couvre certaines des commandes Git les plus courantes que vous pouvez utiliser lors de la gestion des référentiels et de la collaboration sur des logiciels.

Pour en savoir plus sur les logiciels open source et la collaboration, consultez notre série de didacticiels Introduction à l'open source :

- Comment contribuer à l'open source : Premiers pas avec Git
- Comment créer une demande de tirage sur GitHub
- Comment rebaser et mettre à jour une demande de tirage
- Comment gérer des projets de logiciels open source

Il existe de nombreuses autres commandes et variantes que vous pouvez trouver utiles dans le cadre de votre travail avec Git. Pour en savoir plus sur toutes les options disponibles, vous pouvez exécuter la commande suivante pour recevoir des informations utiles :

```
$ git --help
```

Copier

Vous pouvez également en savoir plus sur Git et consulter la documentation de Git sur le site officiel de Git.

En savoir plus sur nous →

Revisitez tous les didacticiels de cette série de didacticiels : Introduction à GitHub et aux projets Open Source →

Série de didacticiels : Introduction à GitHub et aux projets open source

Les projets open source hébergés dans des référentiels publics bénéficient des contributions de la communauté des développeurs au sens large et sont généralement gérés via Git. Cette série de didacticiels vous guidera dans la sélection d'un projet open source auquel contribuer, la création d'une demande de tirage vers un dépôt Git via la ligne de commande et la prise de mesures pour assurer le suivi de votre demande de tirage.

S'inscrire

Git Libre Développement

Parcourir la série : 6 articles

1/6 Comment contribuer à l'open source : Premiers pas avec Git

2/6 Comment créer une pull request sur GitHub

3/6 Comment rebaser et mettre à jour une demande de tirage

Agrandir pour tout afficher

À propos des auteurs

Lisa Tagliaferri Auteur

Vous cherchez toujours une réponse ? Poser une question

Rechercher plus d'aide

Ces informations vous-ont elles été utiles ? Oui Non

Commentaires

Laisser un commentaire

Leave a comment...

This textbox defaults to using `Markdown` to format your answer.

You can type `!ref` in this text area to quickly search our full set of tutorials, documentation & marketplace offerings and insert the link!

Sign In or Sign Up to Comment

Sujets populaires

Ubuntu

Principes de base de Linux

JavaScript (en anglais)

Python

MySQL (en anglais)

Docker

Kubernetes (en anglais)

Tous les tutoriels →

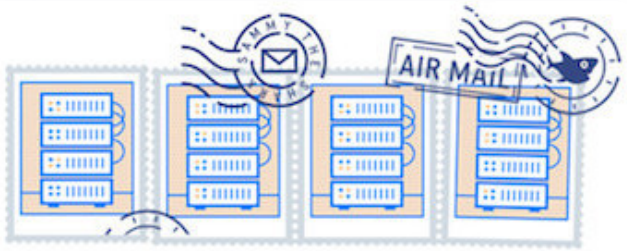
Parlez à un expert →

Congratulations on unlocking the whale ambience easter egg! Click the whale button in the bottom left of your screen to toggle some ambient whale noises while you read.

Thank you to the Glacier Bay National Park & Preserve and Merrick079 for the sounds behind this easter egg.

Interested in whales, protecting them, and their connection to helping prevent climate change? We recommend checking out the Whale and Dolphin Conservation.

Reset easter egg to be discovered again / Permanently dismiss and hide easter egg



Get our biweekly newsletter

Sign up for Infrastructure as a Newsletter.

Sign up →

Hollie's Hub for Good

Working on improving health and education, reducing inequality, and spurring economic growth? We'd like to help.

Learn more →

Become a contributor

You get paid; we donate to tech nonprofits.

Learn more →

Featured on Community

Kubernetes Course | Learn Python 3 | Machine Learning in Python | Getting started with Go | Intro to Kubernetes

DigitalOcean Products

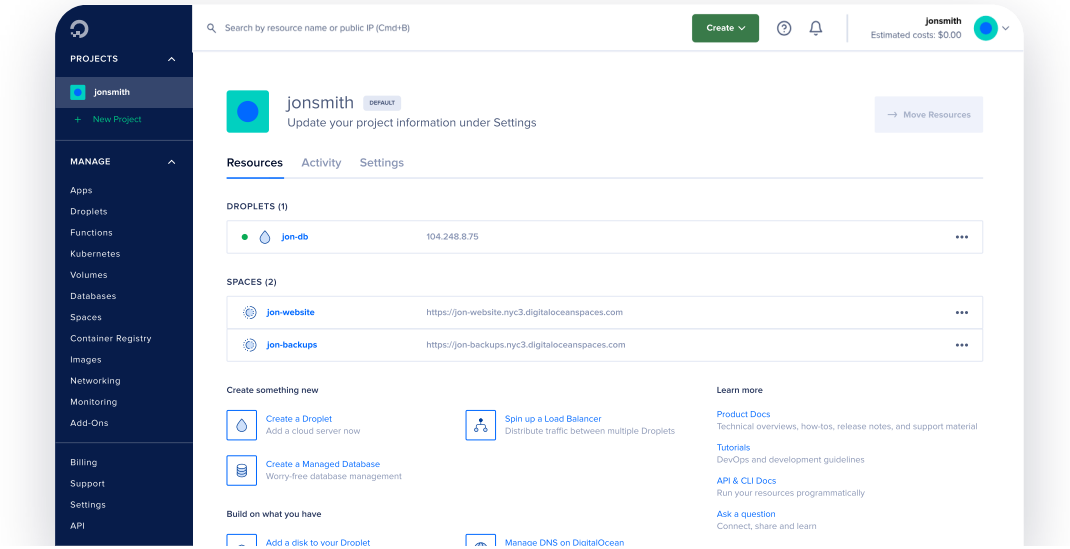
Cloudways | Virtual Machines | Managed Databases | Managed Kubernetes | Block Storage | Object Storage

Marketplace | VPC | Load Balancers

Welcome to the developer cloud

DigitalOcean makes it simple to launch in the cloud and scale up as you grow – whether you're running one virtual machine or ten thousand.

Learn more →



Get started for free

Sign up and get $200 in credit for your first 60 days with DigitalOcean.

Get started

This promotional offer applies to new accounts only.

Release Notes

Uptime

Contact

Support

Sales

Report Abuse

System Status

Share your ideas

 Sitemap.