

Formation Git (Atlassian-Style) : Plan de formation

Table des matières

Formation Git (Atlassian-Style) : Plan de formation

- Table des matières

- Presentation

 - Git

 - Version Control

 - Source Code Management

- Git pour les développeurs

 - Feature Branch Workflow

 - Distributed Development

 - Pull Requests

- Installer, Configurer Git et Github

 - Installer Git

 - Linux

 - OS X

 - Windows

 - Configuration Git et Github

- Cheat Sheet

- Getting Started

 - Basic Git Flow

 - Glossaire

 - working tree / working directory / workspace

 - index / staging area

 - (local) repository

 - remote repository

 - origin

 - commit

 - Cas réel – Créer un projet sur Github

 - Initialiser un repository git

 - Faire un commit

 - Lier le remote au local repository

 - Synchroniser le remote origin

- Travail d'équipe

 - Cloner un repository Github

 - Kit de survie

 - Voir l'état du working tree

 - Afficher l'historique des commits

 - Stocker les changements dans un repository "caché"

 - git commit, en précision

 - Observer les différences entre les versions/fichiers

- Cacher des secrets (ignorer des fichiers/dossiers au git add)
- Gitflow Workflow
- Application : Cas réel
 - Faire des branches
 - Publier les branches
 - Mettre à jour les informations du remote origin
 - Changer de branche
 - Faire une fonctionnalité et la publier
 - Mettre à jour les informations du remote origin et merge origin vers local
 - Merge develop vers feature/my_feature
 - Déployer, de manière Agile, une branche feature vers develop
 - Merge, de manière Agile, une branche feature vers develop
 - Fixer des merge conflicts
 - Merge, de manière Agile, une branche develop vers release
- Googlez les bases du DevOps en 5 secondes
 - WTF is DevOps ?
 - Culture
 - Automation : Enable on demand creation of dev, test, and production environments
 - Lean : Create our single repository of truth for the entire system
 - Monitor & Communication

Presentation

Git

Git est un logiciel de gestion de versions décentralisé. C'est un logiciel libre créé par Linus Torvalds, auteur du noyau Linux, et distribué selon les termes de la licence publique générale GNU version 2. En 2016, il s'agit du logiciel de gestion de versions le plus populaire qui est utilisé par plus de douze millions de personnes.

- Wikipedia

Version Control

Version Control Systems (VCS) have seen great improvements over the past few decades and some are better than others. VCS are sometimes known as SCM (Source Code Management) tools or RCS (Revision Control System). One of the most popular VCS tools in use today is called Git. Git is a *Distributed* VCS, a category known as DVCS, more on that later. [...] The primary benefits you should expect from version control are as follows.

1. A complete long-term change history of every file.
2. Branching and merging.
3. Traceability.

- Atlassian

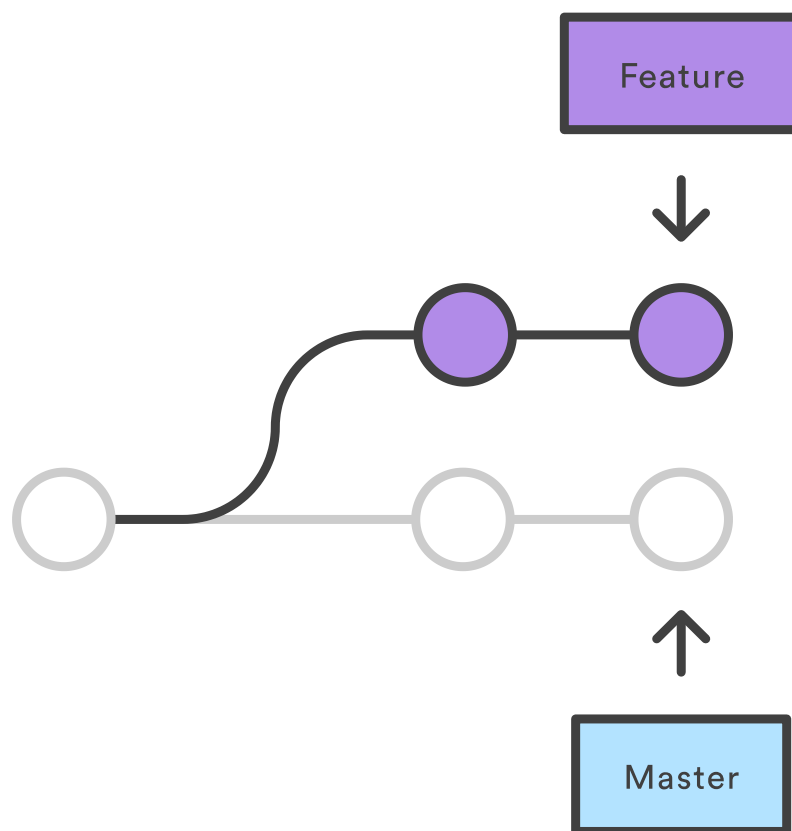
Source Code Management

- Faire des commit régulier
- Être toujours à jour via `git pull`
- Faire des notes détaillés via `git commit`
- Faire des Code Review
- Faire des branches
- Se mettre d'accord sur un workflow

Git pour les développeurs

Feature Branch Workflow

One of the biggest advantages of Git is its branching capabilities. Unlike centralized version control systems, Git branches are cheap and easy to merge. This facilitates the feature branch workflow popular with many Git users.

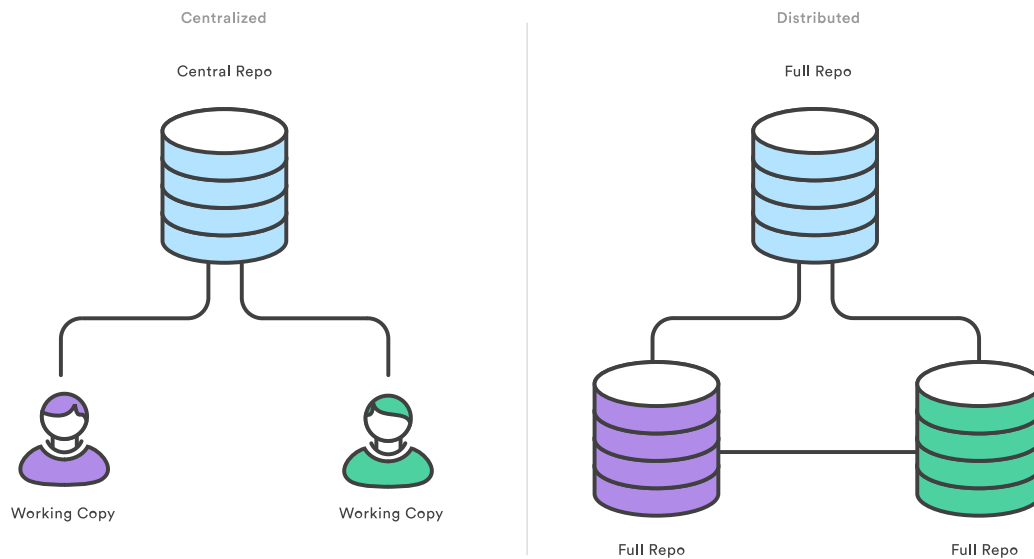


Feature branches **provide an isolated environment for every change to your codebase**. When a developer wants to start working on something—no matter how big or small—they create a new branch. **This ensures that the master branch always contains production-quality code.**

Using feature branches is not only more reliable than directly editing production code, but it also provides **organizational benefits**. They let you **represent development work at the same granularity as the your agile backlog**. For example, you might implement a policy where each Jira ticket is addressed in its own feature branch.

Distributed Development

In SVN, each developer gets a working copy that points back to a single central repository. Git, however, is a **distributed version control system**. Instead of a working copy, **each developer gets their own local repository, complete with a full history of commits**.



Having a full local history makes Git fast, since it means you don't need a network connection to create commits, inspect previous versions of a file, or perform diffs between commits.

Distributed development also makes it **easier to scale your engineering team**. If someone breaks the production branch in SVN, other developers can't check in their changes until it's fixed. With Git, this kind of blocking doesn't exist. Everybody can continue going about their business in their own local repositories.

And, similar to feature branches, distributed development creates a **more reliable environment**. Even if a developer obliterates their own repository, they can simply **clone someone else's and start anew**.

Pull Requests

Many source code management tools such as [Bitbucket](#) enhance core Git functionality with pull requests. A pull request is a way to **ask another developer to merge one of your branches into their repository**. This not only makes it easier for project leads to keep track of changes, but also lets developers initiate discussions around their work before integrating it with the rest of the codebase.


```
brew install git
```

Windows

Git for the Windows platform :

<https://git-scm.com/download/win>

Chocolatey :

```
choco install git
```

Configuration Git et Github

Configurez votre nom :

```
git config --global user.name "John Smith"
```

Configurez votre mail (doit être permanent et servira comme signature, et pourra servir comme signature PGP) :

```
git config --global user.email "john.smith@example.com"
```

Créez une clé SSH ED25519 :




```
$ ssh-keygen -t ed25519
```

```
Generating public/private ed25519 key pair.  
Enter passphrase (empty for no passphrase): [ENTER]  
Enter same passphrase again: [ENTER]  
Your identification has been saved in id_ed25519.  
Your public key has been saved in id_ed25519.pub.  
The key fingerprint is:  
SHA256:[SHA256] description  
The key's randomart image is:  
+--[ED25519 256]--+  
[SHAT256 RANDOMART]  
+-----[SHA256]-----+
```


Exécutez le ssh-agent :

```
eval "$(ssh-agent -s)"
```

Ajoutez la clé **publique** sur votre remote :



Signed in as **octocat**

 Set status

Your profile

Your repositories

Your projects

Your stars

Your gists

Help

Settings

Sign out

Billing


SSH and GPG keys

Security

SSH keys

New SSH key

There are no SSH keys with access to your account.

 Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH Problems](#).

SSH keys

New SSH key


There are no SSH keys with access to your account.

Title

Key

Begins with 'ssh-rsa', 'ssh-dss', 'ssh-ed25519', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', or 'ecdsa-sha2-nistp521'

Add SSH key

 Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH Problems](#).

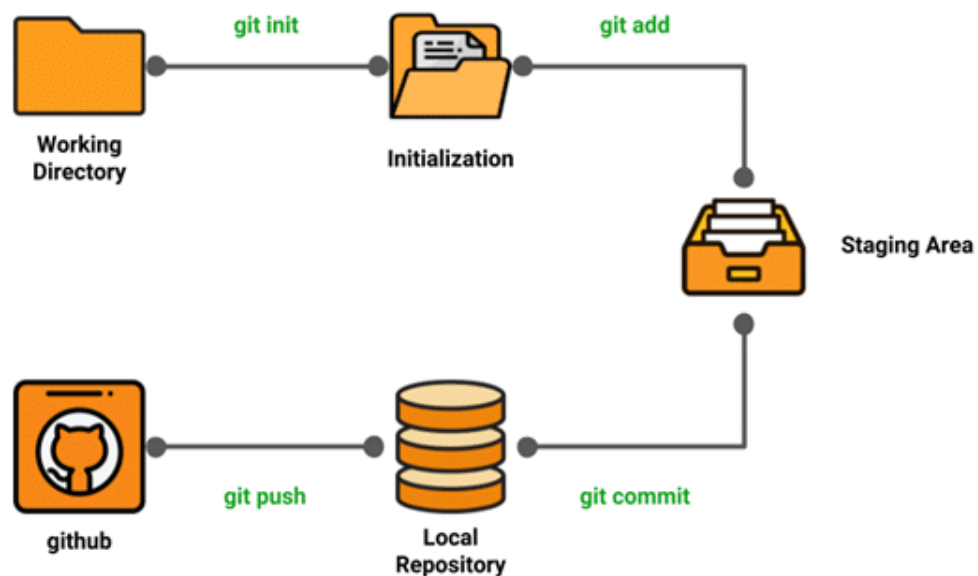
Add SSH key

Cheat Sheet

- [Github Cheat Sheet](#)
- [Atlassian Cheat Sheet](#)
- [Git Tower Cheat Sheet](#)
- [Gitlab Cheat Sheet](#)
- [Roger Dudler's Cheat Sheet](#)

Getting Started

Basic Git Flow



Glossaire

working tree / working directory / workspace

The tree of actual checked out files. The working tree normally contains the contents of the HEAD commit's tree, plus any local changes that you have made but not yet committed.

index / staging area

A collection of files with stat information, whose contents are stored as objects. The index is a **stored version of your working tree**. Truth be told, it can also contain a second, and even a third version of a working tree, which are used when merging.

(local) repository

A collection of refs together with an object database containing all objects which are reachable from the refs, possibly accompanied by meta data from one or more porcelains. A repository can share an object database with other repositories via alternates mechanism.

remote repository

A repository which is used to **track the same project but resides somewhere else**. To communicate with remotes, see fetch or push.

origin

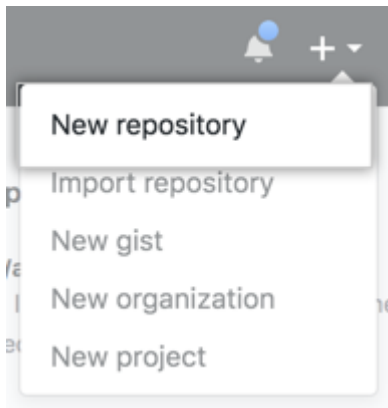
The **default upstream repository**. Most projects have at least one upstream project which they track. By default *origin* is used for that purpose. New upstream updates will be fetched into remote-tracking branches named `origin/name-of-upstream-branch`, which you can see using `git branch -r`.

commit

As a noun: **A single point in the Git history**; the entire history of a project is represented as a set of interrelated commits. The word "commit" is often used by Git in the same places other revision control systems use the words "revision" or "version". Also used as a short hand for commit object.

As a verb: The action of **storing a new snapshot of the project's state in the Git history**, by creating a new commit representing the current state of the index and advancing HEAD to point at the new commit.

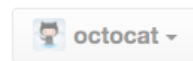
Cas réel - Créer un projet sur Github



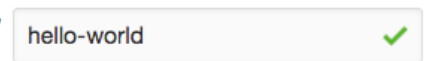
Create a new repository

A repository contains all the files for your project, including the revision history.

Owner



Repository name



Great repository names are short and memorable. Need inspiration? How about **potential-eureka**.

Description (optional)

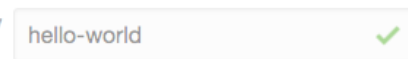
Create a new repository

A repository contains all the files for your project, including the revision history.

Owner



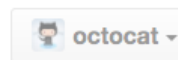
Repository name



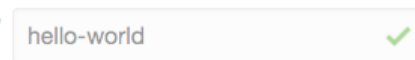
Great repository names are short and memorable. Need inspiration? How about **potential-eureka**.

Description (optional)

Owner





Repository name



Great repository names are short and memorable. Need inspiration? How about **potential-eureka**.

Description (optional)

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

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

Pas de README.

Create repository.

Initialiser un repository git

Initialise un bare repository.

```
git init
```

Faire un commit

Ajouter un fichier/dossier dans le staging area.

```
git add exemple.png  
git add . # Ajoute le dossier actuel
```

Commit :

```
git commit  
git commit -m <message>
```

Lier le remote au local repository

```
git remote add origin git@github.com:auteur/depot.git
```

Synchroniser le remote origin

```
git push
```

Travail d'équipe

Cloner un repository Github

```
git clone git@github.com:auteur/depot.git
```

Kit de survie

Voir l'état du working tree

```
git status
```

Afficher l'historique des commits

```
git log
```

Stocker les changements dans un repository "caché"

Sauvegarde :

```
git stash
```

Lister les sauvegardes :

```
git stash list
```

Charger une sauvegarde :

```
git stash apply [<stash>]
```

Supprimer une sauvegarde :

```
git stash pop [<stash>]
```

git commit, en précision

```
git commit
git commit -a # Commit tous les changements des fichiers suivis
git commit -m <message>
git commit --amend # Refaire le dernier commit
git commit -s/--signoff # Signer avec l'email
git commit -S/--gpg-sign[=<keyid>] # Signer avec une clé GPG
```

Observer les différences entre les versions/fichiers

```
git diff
```

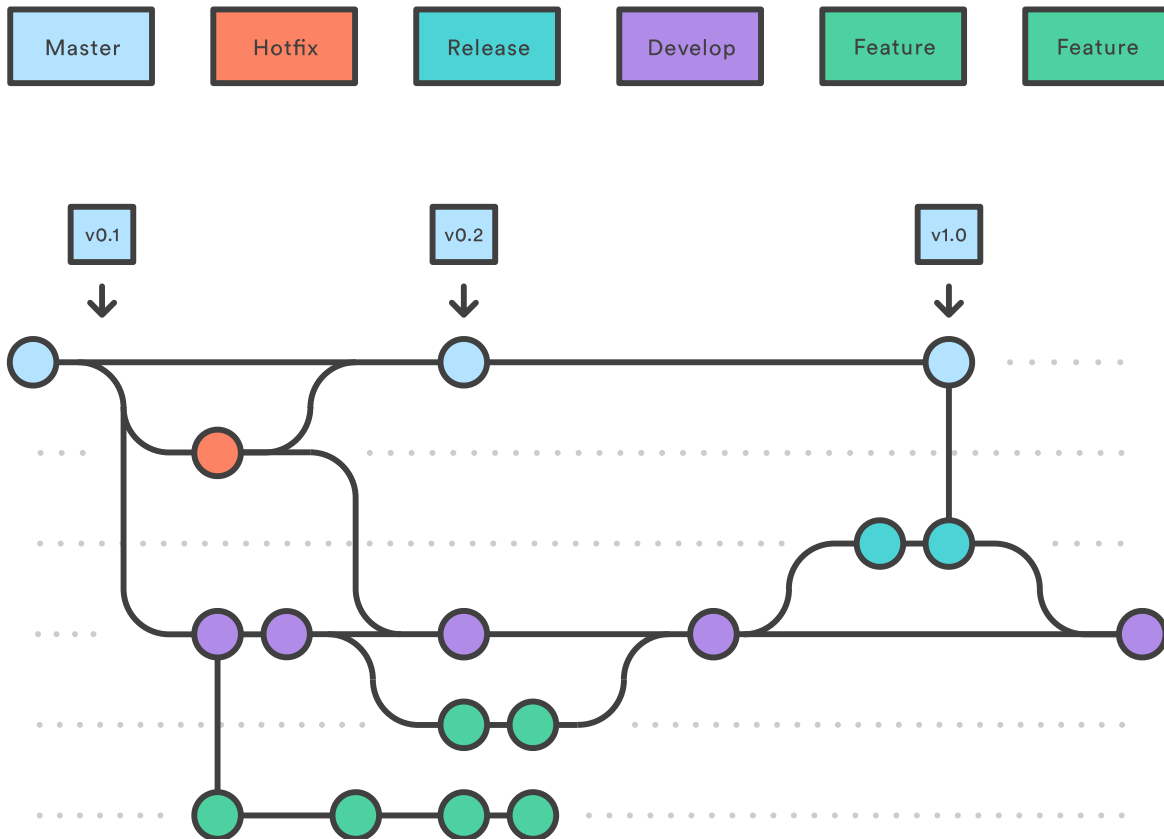
Conseil : Utilisez un bon IDE ([VSCode](#), [Atom](#), ...) ou un bon Git GUI ([GitKraken](#), [Github Desktop](#), ...)

Cacher des secrets (ignorer des fichiers/dossiers au git add)

Créez un fichier .gitignore et écrivez des patterns pour cacher :

```
*/dossier # Tout ce qui est avant dossier
*/fichier.* # Tous les fichiers qui commence par "fichier"
*/dossier/** # Tous les dossiers et sous-dossiers
!/fichier.ext # Sauf fichier.ext
```

Gitflow Workflow



- **master** : État du code source prêt pour la production.
- **develop** : Derniers changements de développement livrés ou aussi appelé branche d'intégration.
- **feature/???** : Fonctionnalité.
- **release/???** : Branche pre-déploiement.
- **hotfix/???** : Littéralement, fix rapide.

Application : Cas réel

Faire des branches

```
git branch develop
git checkout develop
```

ou

```
git checkout -b develop
```

Publier les branches

```
git push -u origin develop
```

Mettre à jour les informations du remote origin

```
git fetch
```

Changer de branche

```
git checkout develop
```

Faire une fonctionnalité et la publier

```
git checkout -b feature/my_feature  
[work.]  
git add .  
git commit -m "My feature init"  
git push -u origin feature/my_feature
```

Mettre à jour les informations du remote origin et merge origin vers local

Assurez-vous d'être sur develop :

```
git checkout develop
```

Maintenant, faites merge master:

```
git fetch # Mettre à jour au cas où  
git merge origin/develop
```

ou

```
git pull
```

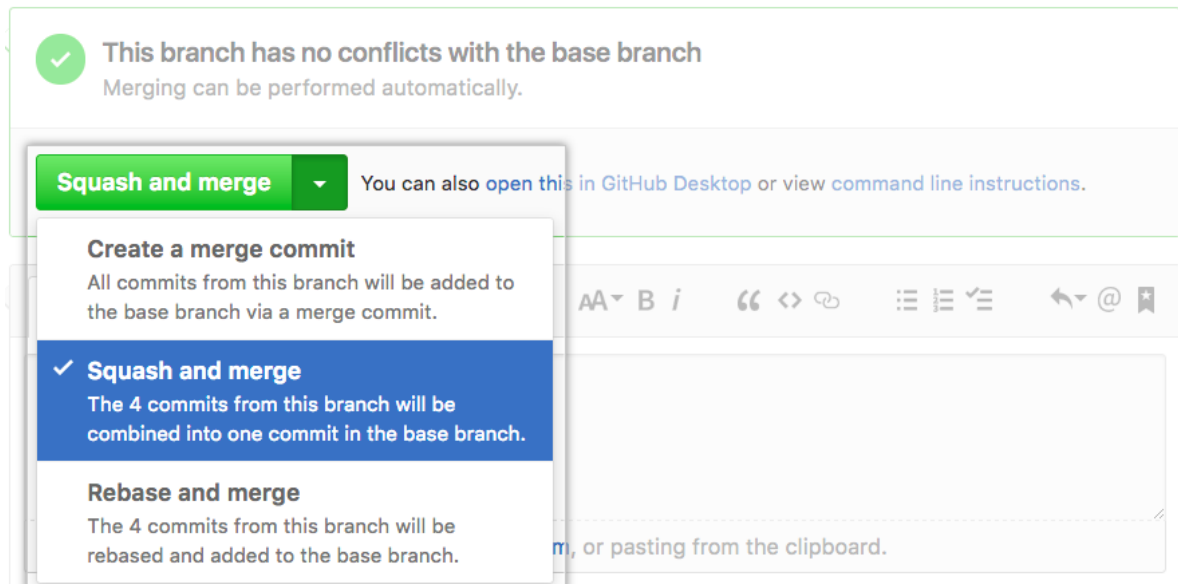
Merge develop vers feature/my_feature

```
git checkout feature/my_feature  
git merge develop  
git push # Publiez (ou pas)
```

Déployer, de manière Agile, une branche feature vers develop

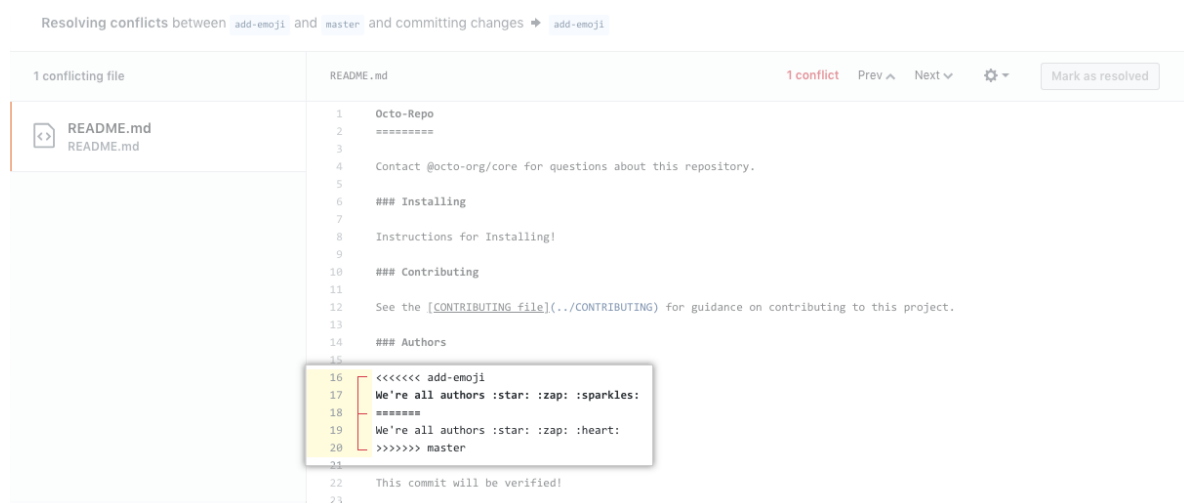
Aller sur Github et demandez un Pull Request.

Merge, de manière Agile, une branche feature vers develop



Fixer des merge conflicts

A la main, choisir :

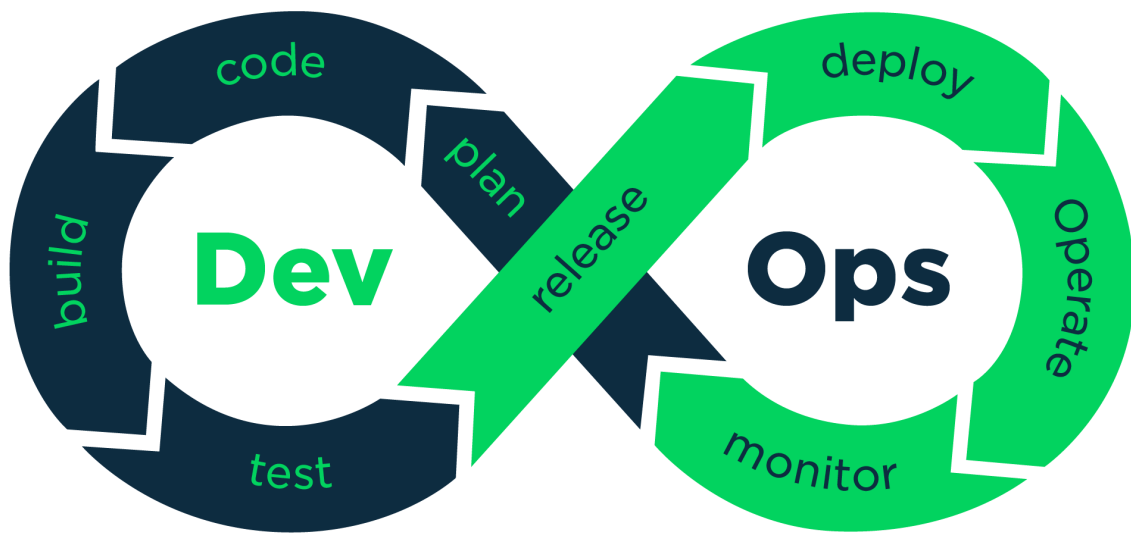


Sinon, un bon IDE ([VSCode](#), [Atom](#), ...) ou un bon Git GUI ([GitKraken](#), [Github Desktop](#), ...).

Merge, de manière Agile, une branche develop vers release



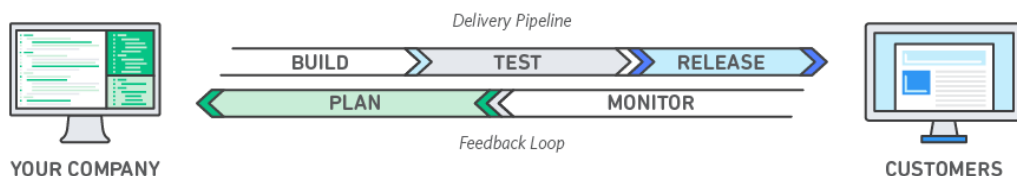
Googlez les bases du DevOps en 5 secondes



WTF is DevOps ?

D'après [AWS Amazon](#),

DevOps est une combinaison de **philosophies culturelles**, de **pratiques** et d'**outils** qui **améliore** la capacité d'une entreprise à **livrer des applications et des services à un rythme élevé**. Il permet de faire **évoluer et d'optimiser les produits plus rapidement** que les entreprises utilisant des processus traditionnels de développement de logiciels et de gestion de l'infrastructure. Cette vitesse permet aux entreprises de **mieux servir leurs clients et de gagner en compétitivité**.



Dans un modèle DevOps, **les équipes de développement et d'opérations ne sont plus isolées**. Il arrive qu'elles soient **fusionnées en une seule et même équipe**. Les ingénieurs qui la composent travaillent alors sur tout le cycle de vie d'une application, de la création à l'exploitation, en passant par les tests et le déploiement, et développent toute une gamme de compétences liées à différentes fonctions.

Dans certains modèles DevOps, les équipes **d'assurance qualité** et de **sécurité** peuvent également s'intégrer étroitement au développement et aux opérations, ainsi qu'à l'ensemble du cycle de vie des applications. Lorsque la sécurité est au cœur de l'activité d'une équipe DevOps, on parle parfois de **DevSecOps**.

Ces équipes utilisent des pratiques pour automatiser des processus qui étaient autrefois manuels et lents. Elles exploitent une **pile technologique et des outils** qui les aident à faire fonctionner et à faire évoluer les applications de façon rapide et fiable. Ces outils aident également les ingénieurs à **accomplir de façon autonome des tâches** (par exemple, le déploiement de code ou la mise en service

d'infrastructure) qui nécessiteraient normalement l'aide d'autres équipes, **ce qui augmente encore davantage la productivité de l'équipe d'ingénieurs.**

Culture

Automation : Enable on demand creation of dev, test, and production environments

- Export d'environnement virtualisé : [Vagrant](#), [VMware Image](#), ...
- **Preboot Execution Environment** : [Debian PXEBootInstall](#), ...
- **"Infrastructure as Code" (IaC) configuration management** : [Puppet](#), [Ansible](#), [Chef](#), ...
- Configuration d'OS automatisée : [Debian preseed](#), [Red Hat Kickstart](#), ...
- Approvisionnement automatisé des VM ou des Containers : [Vagrant](#), [Docker](#), ..
- Mise en place automatisée un environnement sur un cloud public : [Google App Engine](#), [Microsoft Azure](#), [Amazon Web Services](#), ...
- Mise en place automatisée un environnement sur un cloud privé : Do it yourself !
- Mise en place automatisée un environnement sur un "Platform as a Service" (PaaS) : [OpenStack](#), [CloudFoundry](#), ...

Lean : Create our single repository of truth for the entire system

- **Version Control System** : [Git](#), [SVN](#), [CVS](#), [Bazaar](#), [Mercurial](#)
- **Git hosting** : [Github](#), [Gitlab](#), [Bitbucket](#), [SourceForge](#), [Launchpad](#), DIY, ...
- **Continuous Integration and Continuous Delivery (CI/CD)** : [Github Actions](#), [Gitlab CI](#), [Travis CI](#), [Appveyor](#), [Google Cloud Build](#)...
 - Build
 - Make
 - CMake
 - Ninja
 - Gradle
 - Test
 - Unit Tests (UT)
 - Integration Tests (IT)
 - GUI Tests
 - Static Application Security Testing (SAST)
 - Dynamic Application Security Testing (DAST)
 - Interactive Application Security Testing (IAST)
 - Exploratory Testing (manual)
 - User Acceptance Testing (UAT) (manual)
 - Production Testing (manual)
 - Deploy (**FULL AUTOMATED**)
- **Microservices** : [AWS Lambda](#), [Google App Engine](#) et pratiquement tout ce qu'il y a dans Automation.

Monitor & Communication

- **Atlassian & Jira** (Project, Issue Tracking, Agile Planning, Management, Service Desk, Customer Service, Incident Management, Incident

Communication)

- **Github** (Project, Issue Tracking, Agile Planning, Kanban, Milestones)
- Et plein d'autres...