



**Alphorm**.com

# Formation Découverte **Git**

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>



**Édouard FERRARI**  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

Formation Git

alphorm.com™©

## Plan

- Présentation du formateur
- Plan de formation
- Objectifs de la formation
- Public concerné
- Les possibilités de Git
- Les connaissances requises



Formation Git

alphorm.com™©

## Présentation du formateur

Edouard FERRARI



- [contact@ferrari.wf](mailto:contact@ferrari.wf)
- Développeur full stack chez Summview
- Mission de conseil, d'architecture et de migration
- Mes références :
  - LinkedIn : <https://fr.linkedin.com/in/edouardferrari>
  - Alphorm : <http://www.alphorm.com/formateur/edouard-ferrari>
  - Github : <https://github.com/didouard>

Formation Git

alphorm.com™©

## Mes formations

- Coursus complet sur NodeJS

Formation NodeJS, les fondamentaux

Apprendre les bases de NodeJS, votre chemin moderne vers des applications web avec une montée en charge importante

★★★★★ (5votes), 14640vues

  Formation Les fondamentaux de NodeJS

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

Édouard FERRARI  
Formateur et Consultant indépendant  
Contact : [contact@ferrari.wf](mailto:contact@ferrari.wf)

39€ 19.5€  
Promo valable jusqu'à Dimanche 3 inclus...

ACHETEZ LA FORMATION A VIE

S'ABONNER à partir de 15€  
Voir les vidéos gratuites

- ✓ Formation vidéo de 9h35min44s
- ✓ Satisfait ou remboursé
- ✓ Accès à vie et visionnage illimité
- ✓ Attestation de fin de formation
- ✓ Conçu par un formateur expert
- ✓ Consultable sur iOS/Android
- ✓ Fichiers sources inclus

Formation NodeJS, avancé

Maîtrisez des fonctionnalités et modules avancés sous NodeJS tout en construisant une application complète de chat comme Whatsapp.

★★★★★ (5votes), 4323vues

  Formation NodeJS Avancé

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

Édouard FERRARI  
Formateur et Consultant indépendant  
Contact : [contact@ferrari.wf](mailto:contact@ferrari.wf)

49€ 24.5€  
Promo valable jusqu'à Dimanche 3 inclus...

ACHETEZ LA FORMATION A VIE

S'ABONNER à partir de 15€  
Voir les vidéos gratuites

- ✓ Formation vidéo de 11h27min11s
- ✓ Satisfait ou remboursé
- ✓ Accès à vie et visionnage illimité
- ✓ Attestation de fin de formation
- ✓ Conçu par un formateur expert
- ✓ Consultable sur iOS/Android
- ✓ Fichiers sources inclus

Formation Git

alphorm.com™©



## Plan de la formation

- **Présentation de Git**
  - Gestion des versions
  - Rudiments de Git
  - Installation de Git
  - Paramétrage de Git
- **Les bases de Git**
  - Démarrer un dépôt Git
  - Enregistrer des modifications dans le dépôt
  - Visualiser l'historique des validations
  - Annuler des actions
  - Travailler avec des dépôts distants
  - Étiquetage
- **Les branches avec Git**
  - Les branches en bref
  - Branches et fusions : les bases
  - Gestion des branches
  - Travailler avec les branches
  - Branches distantes
  - Rebaser (Rebasing)
- **Github**
  - Configuration et paramétrage d'un compte
  - Contribution à un projet
  - "Github flavored Markdown"
  - Maintenance d'un projet
  - Gestion d'un regroupement
  - Écriture de scripts pour GitHub



## Objectifs de la formation

- Étudier Git pour pouvoir l'utiliser pour des projets personnels ou professionnels.
- Git surclasse les autres outils **SCM** (*Source Code Management*) par sa performance, la taille des dépôts et ses fonctionnalités uniques.
- Dans cette formation nous allons étudier en profondeur les fondamentaux de Git. Bien comprendre les bases, pour voir des notions plus complexes lors de **la seconde formation**.
- A l'issue de la formation, vous aurez appris à configurer et utiliser GIT dans un contexte de gestion quotidienne des sources d'un projet professionnel.

## **Public concerné**

---

- À qui s'adresse cette formation :
  - Aux étudiants
  - Aux développeurs
  - Aux chefs de projet
  - Aux amoureux des nouvelles technologies
  - Aux personnes souhaitant mieux s'organiser
- A tout le monde !

## **Les possibilités de Git**

---

- Git permet :
  - De versionner ses fichiers (code, images, documents, ...)
  - De mettre en attente une version et de travailler sur une autre
  - De pouvoir fusionner un même fichier sur lequel plusieurs personnes ont travaillé
  - Organiser son travail par version
  - Publier en production son code à partir d'une version donnée
  - Avoir une historique précise de son projet
  - De pouvoir blâmé quelqu'un ! ;-)
  - ...



## Les connaissances requises

---

- Aucune !

## Liens et ressources

---

- <https://git-scm.com/> : Site officiel de Git
- [Http://github.com](http://github.com) : Plateforme en ligne de Git
- <http://gitref.org/> : Pour ne jamais oublier les commandes
- <https://try.github.io/> : Pour s'entraîner

## Let's Git 😊



Formation Git

alphorm.com™©



**Alphorm**.com

## Présentation de GIT Gestion des versions

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>



**Édouard FERRARI**  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

Formation Git

alphorm.com™©

## Plan

---

- Qu'est-ce qu'un gestionnaire de version ?
- Les systèmes de gestion de version local
- Les systèmes de gestion de version centralisé
- Les systèmes de gestion de version distribué

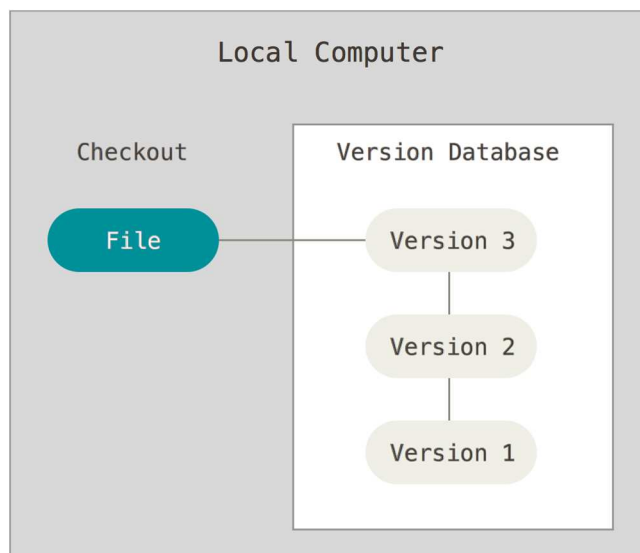


## Qu'est-ce qu'un gestionnaire de version ?

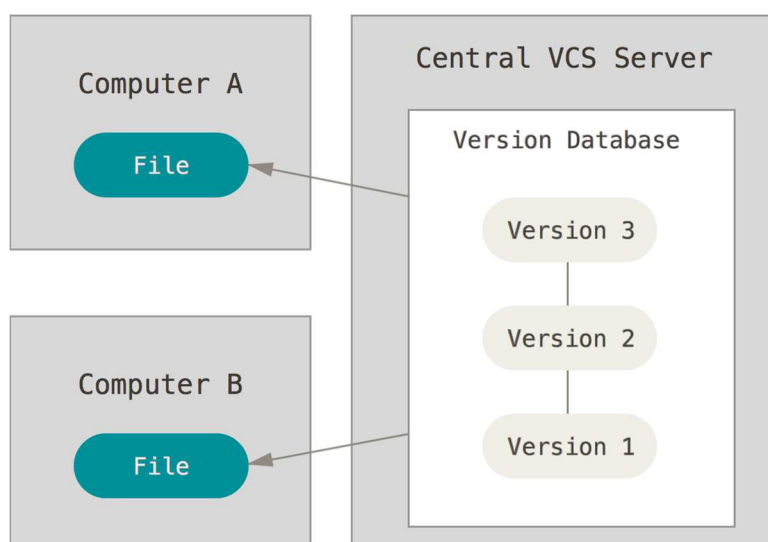
---

- Un gestionnaire de version :
  - Enregistre les évolutions d'un fichier
  - Permet de revenir à une version précédente
  - Fonctionne avec tout type de fichier (.txt, .php, .java, .jpg, .exe, ...)
  - Permet de retrouver un fichier supprimé

## Les systèmes de gestion de version locaux

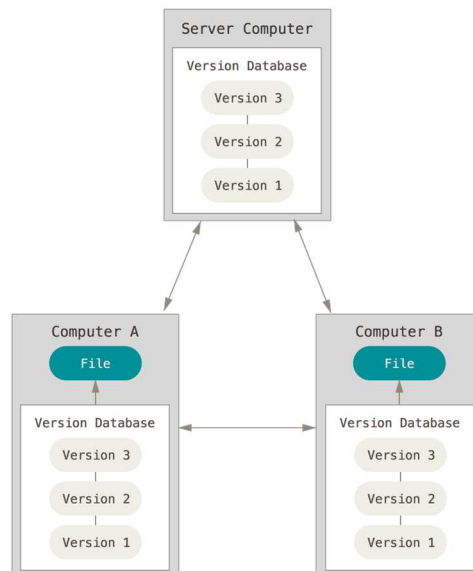


## Les systèmes de gestion de version centralisés





## Les systèmes de gestion de version distribués



## Ce qu'on a couvert

- Nous avons vu dans cette vidéo 3 grandes méthodes de gestion des versions pour nos fichiers.
- Git utilise la 3<sup>ème</sup> solution !
- La prochaine vidéo :
  - Les rudiments de GIT





**Alphorm**.com

## Présentation de GIT

### Rudiments de GIT

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>



**Édouard FERRARI**  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

Formation Git

alphorm.com™©

## Plan

- L'histoire de GIT
- Des instantanés, pas des différences
- Presque toutes les opérations sont locales
- Git gère l'intégrité
- Les trois états



Formation Git

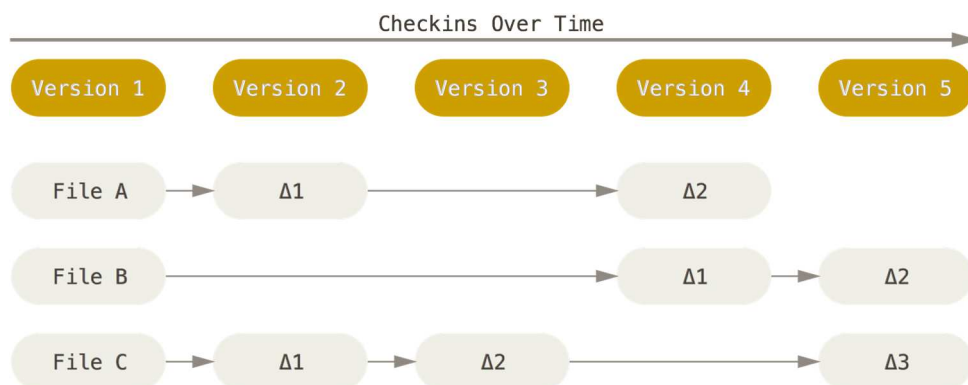
alphorm.com™©

## L'histoire de GIT

- Git est né avec une dose de destruction créative et de controverse houleuse.
- Créé par **Linus Trovalds**, le créateur de Linux en 2005.
- Nouveaux objectifs:
  - Vitesse
  - Conception simple
  - Support pour les développements non linéaires
  - Complètement distribué
  - Capacité à gérer efficacement des projets d'envergure tels que le noyau Linux

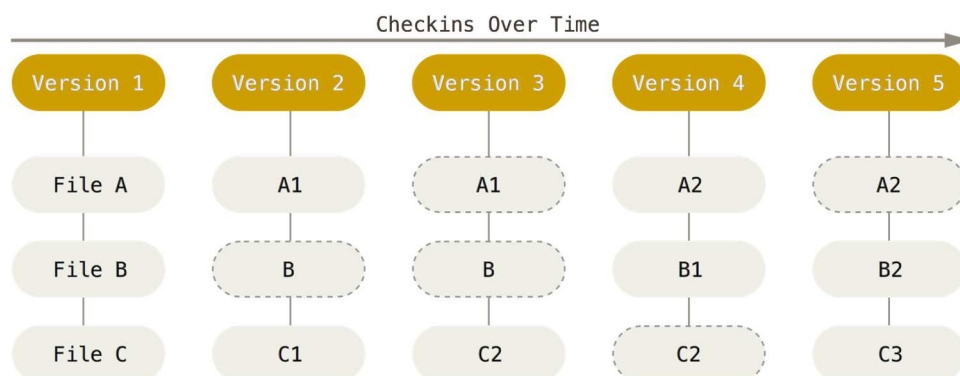
## Des instantanés, pas des différences

- Beaucoup de VCS considèrent l'information qu'ils gèrent comme une liste de fichiers et les modifications effectuées sur chaque fichier dans le temps.



## Des instantanés, pas des différences

- Avec **Git**, à chaque fois que vous validez ou enregistrez l'état du projet, il prend un instantané du contenu de votre espace de travail.



## Presque toutes les opérations sont locales

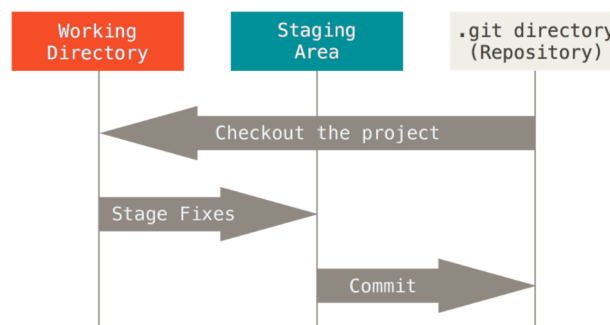
- La plupart des opérations de Git se font en local.
- Même pour chercher dans l'historique du projet.
- L'intégralité de la base de données est en **locale**.

## Git gère l'intégrité

- Dans Git, tout est vérifié par une somme de contrôle avant d'être stocké et par la suite cette somme de contrôle, signature unique, sert de référence.
- Impossible de perdre un fichier dans Git sans que Git s'en aperçoive.
- Les sommes de contrôle sont en **SHA-1**.
  - 24b9da6552252987aa493b52f8696cd6d3b00373

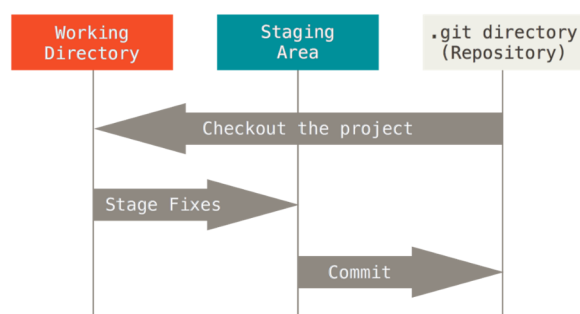
## Les trois états

- Un fichier versionné sous Git peut être sous **3 états** :
  - **Validé**, le fichier est sauvegardé en base.
  - **Modifié**, le fichier est modifié, la modification n'est pas en base.
  - **Indexé**, le fichier est modifié, et est prêt à être envoyé en base.



## Les trois états

- Ce qui nous donne trois sections principales d'un projet GIT
  - **Validé** = le répertoire `.git`
  - **Modifié** = Le répertoire de travail courant
  - **Indexé** = La zone virtuelle qui liste les fichiers qui seront dans le prochain instantané



## Ce qu'on a couvert

- Nous avons vu dans cette vidéo :
  - Comment Git s'est créée.
  - Comment git a été conçu.
  - Comment git fonctionne.
- La prochaine vidéo :
  - Installation de GIT





**Alphorm**.com

## Présentation de Git

---

## Installation de Git

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>



**Édouard FERRARI**  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

Formation Git

alphorm.com™©

### Plan

---

- Installation sous Linux
- Installation sous MacOS
- Installation sous Windows
- Installation à partir de source



Formation Git

alphorm.com™©

## Installation sous Linux

- Généralement, présent sur paquet de votre distribution sous le nom de 'git'
- Sur fedorat : `dnf install git-all`
- Sur Debian / Ubuntu : `apt-get install git-all`
- La liste des distributions est disponible à cette adresse :
  - <https://git-scm.com/download/linux>

## Installation sous MacOS

- Il existe plusieurs méthodes d'installation de Git sur un Mac. La plus facile est probablement d'installer les *Xcode Command Line Tools*.
- Si vous voulez une version plus récente :
  - <https://git-scm.com/download/mac>





## Installation sous Windows

---

- Pour Windows :
  - <http://git-scm.com/download/win>

## Installation depuis les sources

---

- Pour installer Git, vous avez besoin des bibliothèques suivantes :
  - curl, zlib, openssl, expat, libiconv.
- Pour Linux :
  - `apt-get install libcurl4-gnutls-dev libexpat1-dev gettext libz-dev libssl-dev autoconf`
- Les sources sont disponibles sur ce site (miroir) :
  - <https://github.com/git/git/releases>



## Installation depuis les sources

---

- Procédure d'installation :

```
$ tar -zxf git-1.9.1.tar.gz
$ cd git-1.9.1
$ make configure
$ ./configure --prefix=/usr
$ make all doc info
$ sudo make install install-doc install-html install-info
```



## Ce qu'on a couvert

---

- Nous avons vu dans cette vidéo comment installer Git pour
  - Linux, MacOS, Windows
  - À partir des sources
- La prochaine vidéo :
  - Paramétrage de Git





**Alphorm**.com

## Présentation de Git

---

## Paramétrage de Git

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>



**Édouard FERRARI**  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

Formation Git

alphorm.com™©

### Plan

---

- Les fichiers de configuration
- Votre identité
- Votre éditeur de texte
- Vos paramètres



Formation Git

alphorm.com™©



## Les fichiers de configuration

---

- Git contient un outils '**git config**' pour modifier les paramètres de configuration
- Git a 3 niveaux de configuration
  - Global au système : `/etc/gitconfig`, modifiable avec '**git config -- system**'
  - Global à l'utilisateur : `~/.gitconfig`, modifiable avec '**git config --global**'
  - Global au projet : `.git/config`, modifiable avec '**git config**'
- Chaque niveau surcharge le précédent



## Votre identité

---

- La première chose à faire après l'installation de Git est de renseigner votre nom et votre adresse email.  

```
$ git config --global user.name "Edouard FERRARI"
```

```
$ git config --global user.email edouard@ferrari.wf
```
- Vu que nous configurons ces options avec `--global`, nous n'aurons pas besoin de les reconfigurer à chaque projet.

## Votre éditeur de texte

---

- Nous faisons la même chose avec notre éditeur de texte.

```
$ git config --global core.editor emacs
```

## Vos paramètres

---

- Il est possible de vérifier nos paramètres :

```
$ git config --list
```

## Ce qu'on a couvert

---

- Nous avons vu dans cette vidéo comment paramétrer Git
- Le prochain chapitre :
  - Les bases de Git => Démarrer un dépôt Git



Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

**Alphorm**.com

Les bases de Git

---

Démarrer  
un dépôt Git



**Édouard FERRARI**  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

## Plan

---

- Initialisation d'un dépôt Git
- Cloner un dépôt existant



## Initialisation d'un dépôt Git

---

- Dans votre projet, initialiser Git:  

```
$ git init
```
- Un dossier **.git** apparait, l'initialisation est faite, mais aucun fichier n'est versionné !

## Cloner un dépôt existant

---

- La commande pour récupérer un projet existant est 'git clone'.
- Git reçoit une copie de quasiment **toutes** les données stockées dans le serveur.
- S'il y a un problème avec le disque du serveur, vous aurez une copie complète du projet (fichier et historique).
- Git gère le protocole http:// ou https://
- Mais aussi le protocole git://server.tld/projet.git. Ce protocole se base sur ssh.
- Exemple :
  - \$> git clone https://github.com/libgit2/libgit2

## Ce qu'on a couvert

---

- Nous avons vu dans cette vidéo comment :
  - Initialiser un projet
  - Cloner un project existant
- La prochaine vidéo :
  - Enregistrer des modifications dans le dépôt







**Alphorm**.com

## Les bases de Git

### Enregistrer des modifications dans le dépôt

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>



**Édouard FERRARI**  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

Formation Git

alphorm.com™©

## Plan

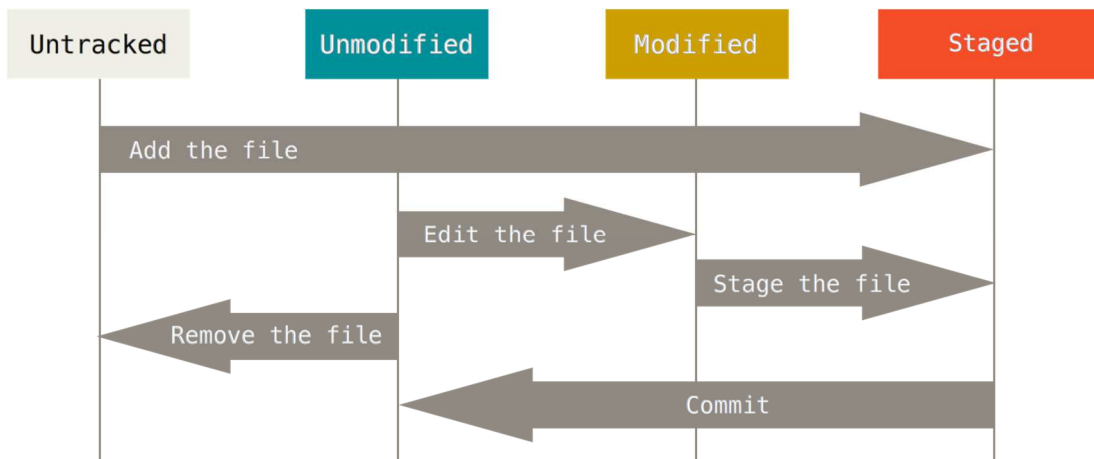
- Le fonctionnement
- Vérifier l'état des fichiers
- Placer de nouveaux fichiers sous suivi de version
- Indexer des fichiers modifiés
- Ignorer des fichiers
- Inspecter les modifications indexées et non indexées
- Valider vos modifications
- Passer l'étape de mise en index
- Effacer ou déplacer des fichiers



Formation Git

alphorm.com™©

## Le fonctionnement



## Vérifier l'état des fichiers

- Pour connaître l'état des fichiers de votre projet :

**\$ git status**

```
$ git status
Sur la branche master
Votre branche est à jour avec 'origin/master'.
rien à valider, la copie de travail est propre
```

- Si on ajoute un nouveau fichier dans le projet, **git status** nous donne son nouvel état.
- Pour avoir un affichage plus concis :

**\$ git status -s**



## Placer de nouveaux fichiers sous suivi de version

- Pour commencer à suivre un nouveau fichier, il faut utiliser la commande :  
`$ git add <fichier>`
- En faisant un '**git status**', nous voyons que le fichier a changé d'état, le fichier est indexé dans Git.
- La version du fichier à l'instant où vous l'enregistrez avec '**git add**' sera celle qui sera dans l'historique des instantanés.
- On peut ajouter des répertoires, git ajoutera à l'index tout le répertoire récursivement.



## Indexer des fichiers modifiés

- Maintenant, modifions un fichier qui est déjà sous suivi de version.
- Puis observons le status avec '**git status**'.
- Le fichier modifié a le status "**Modifications qui ne seront pas validées**", le fichier n'est pas indexé.
- Pour indexer le fichier :  
`$ git add <fichier>`

## Ignorer des fichiers

---

- Dans les projets, nous avons souvent des fichiers que nous ne voulons pas ajouter à Git.
- Nous avons la possibilité de ne jamais les indexer avec '**git add**'.
- Sinon, nous pouvons indiquer à Git de les ignorer.
- Il faut rentrer la liste des fichiers dans le fichier **.gitignore**
- Les fichiers n'apparaîtront plus dans '**git status**'.
  - \$ cat .gitignore
  - \*.oa
  - \*~

## Ignorer des fichiers

---

- Les règles de construction des patrons à placer dans le fichier **.gitignore** sont les suivantes :
  - Les lignes vides ou commençant par # sont ignorées.
  - Les patrons standards de fichiers sont utilisables.
  - Si le patron se termine par une barre oblique (/), il indique un répertoire.
  - Un patron commençant par un point d'exclamation (!) indique des fichiers à inclure malgré les autres règles.
- Exemple de fichiers gitignore :
  - <https://github.com/github/gitignore>



## Inspecter les modifications indexées et non indexées

- Pour avoir une vision plus précise des changements faits sur les fichiers, on utilise '**git diff**'.
- Cette commande répond aux questions :
  - Qu'est-ce qui a été modifié, mais pas encore indexé ?
  - Quelle modification a été indexée et est prête pour la validation ?
- **git diff** montre les changements ligne par ligne
- Pour voir les modifications qui font partie de la prochaine validation, vous pouvez utiliser '**git diff --cached**'



## Valider vos modifications

- Une fois que votre index est dans l'état désiré, nous pouvons valider les modifications.
- Rappel, ce qui n'est pas passé dans '**git add**' ne fera pas partie de la prochaine validation. Ils resteront en modifier sur votre disque.
- La commande pour valider est '**git commit**'
- Cette action lance votre éditeur de texte par défaut.
- Les options :
  - **-v** : ajoute le diff dans le commentaire
  - **-m 'my comment'** : Pour ajouter directement un commentaire sans passer par l'éditeur de texte.
- Une fois votre validation faite, votre commit a un id (SHA-1)

## Passer l'étape de mise en index

- La gestion de la zone d'index peut être longue si on ajoute tous les fichiers un par un.
- **git commit -a** dit à git de placer tous les fichiers déjà suivis dans la zone d'index.

## Effacer ou déplacer des fichiers

- Pour effacer un fichier de Git, vous devez l'éliminer des fichiers en suivi de version (plus précisément, l'effacer dans la zone d'index) puis valider.
- Si vous effacez le fichier avec '**rm <fichier>**', il apparaît sous la section « **Modifications qui ne seront pas validées** » (c'est-à-dire, non indexé).
- Avec un '**git rm <fichier>**', l'effacement du fichier est indexé.



## Effacer ou déplacer des fichiers

- À la différence des autres VCS, Git ne suit pas explicitement les mouvements des fichiers, même s'il sera capable de voir qu'un fichier a été déplacé.
- Il existe la commande '**git mv <source> <destination>**' qui revient exactement à :  

```
$> mv <source> <destination>
```

```
$> git rm <source>
```

```
$> git add <destination>
```



## Ce qu'on a couvert

- Nous avons vu dans cette vidéo :
  - Comment ajouter un fichier à l'index
  - Comment indiquer à git que le fichier déjà suivi a été modifié et qu'on veut l'ajouter à l'index
  - Comment créer un 'commit'
  - Comment supprimer ou déplacer des fichiers.
- Prochaine vidéo :
  - Visualiser l'historique des validations





**Alphorm**.com

## Les bases de Git

---

### Visualiser l'historique des validations

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>



**Édouard FERRARI**  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

Formation Git

alphorm.com™©

## Plan

---

- Visualiser l'historique
- Les options principales
- L'option 'pretty'
- Limiter la longueur de l'historique



Formation Git

alphorm.com™©



## Visualiser l'historique

- Pour voir l'historique, il existe la commande 'git log'
- Pour tester :
  - `$> git clone --branch FR_319_03_02 https://github.com/didouard/project-alphorm project-alphorm-FR_319_03_02`
- Par défaut, **git log** invoqué sans argument énumère en ordre chronologique inversé les commits réalisés.

## Les options principales

- **git log** dispose d'un très grand nombre d'options permettant de paramétrer exactement ce que l'on cherche à voir :

<code>-p</code>	Affiche le patch appliqué par chaque commit
<code>--stat</code>	Affiche les statistiques de chaque fichier pour chaque commit
<code>--shortstat</code>	N'affiche que les ligne modifiées/insérées/effacées de l'option <code>--stat</code>
<code>--name-only</code>	Affiche la liste des fichiers modifiés après les informations du commit
<code>--name-status</code>	Affiche la liste des fichiers affectés accompagnés des informations d'ajout/modification/suppression
<code>--abbrev-commit</code>	N'affiche que les premiers caractères de la somme de contrôle SHA-1
<code>--relative-date</code>	Affiche la date en format relatif (par exemple "2 weeks ago" : il y a deux semaines) au lieu du format de date complet
<code>--graph</code>	Affiche en caractères ASCII le graphe de branches et fusions en vis-à-vis de l'historique
<code>--pretty</code>	Affiche les <i>commits</i> dans un format alternatif. Les formats incluent <code>oneline</code> , <code>short</code> , <code>full</code> , <code>fuller</code> , et <code>format</code> (où on peut spécifier son propre format)
<code>--oneline</code>	Option de convenance correspondant à <code>--pretty=oneline --abbrev-commit</code>

## L'option 'pretty'

- L'option `--pretty` permet d'avoir des affichages différents :

\$> `git log --pretty=oneline` : Affichage d'un commit par ligne

\$> `git log --pretty=format:'...'` : Permet de créer un format spécifique

%H	Somme de contrôle du commit
%h	Somme de contrôle abrégée du commit
%T	Somme de contrôle de l'arborescence
%t	Somme de contrôle abrégée de l'arborescence
%P	Sommes de contrôle des parents
%p	Sommes de contrôle abrégées des parents
%an	Nom de l'auteur
%ae	E-mail de l'auteur
%ad	Date de l'auteur (au format de l'option -date=)
%ar	Date relative de l'auteur
%cn	Nom du validateur
%ce	E-mail du validateur
%cd	Date du validateur
%cr	Date relative du validateur
%s	Sujet

## Ce qu'on a couvert

- Nous avons vu dans cette vidéo :
  - Comment afficher l'historique de notre projet
  - Jouer avec le formatage
- Prochaine vidéo :
  - Annuler des actions





**Alphorm**.com

## Les bases de Git

### Annuler des actions

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>



Édouard FERRARI  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

Formation Git

alphorm.com™©

## Plan

- Modifier un commit
- Désindexer un fichier déjà indexé
- Réinitialiser un fichier modifié




Formation Git

alphorm.com™©

## Modifier un commit

---

-  Attention, certaines modifications sont permanentes.
- Il peut arriver que l'on valide une modification trop tôt en oubliant des fichiers.
- Pour modifier un commit :  

```
$> git commit --amend
```
- Cette commande va reprendre l'index du commit précédent.
- L'éditeur s'ouvre avec les modifications et le message.
- Il est possible de modifier uniquement le message du commit.

## Désindexer un fichier déjà indexé

---

- Imaginons que vous avez indexé trop de fichiers.  

```
$> git reset HEAD <file>
```



## Réinitialiser un fichier modifié

- Pour faire revenir un fichier au niveau du précédent checkout :  
`$> git checkout -- <fichier>`
- Attention, vous perdrez définitivement les modifications apportées au fichier depuis le dernier checkout.
- Si vous souhaitez seulement écarter momentanément cette modification, nous verrons comment mettre de côté et créer des branches



## Ce qu'on a couvert

- Nous avons vu dans cette vidéo :
  - Comment modifier un commit.
  - Comment désindexer un fichier déjà indexé.
  - Comment réinitialiser un fichier modifié.
- La prochaine vidéo :
  - Travailler avec des dépôts distants





**Alphorm**.com

## Les bases de Git

---

### Travailler avec des dépôts distants

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>



**Édouard FERRARI**  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

Formation Git

alphorm.com™©

## Plan

---

- Introduction
- Afficher les dépôts distants
- Afficher les dépôts distants
- Récupérer et tirer depuis des dépôts distants
- Pousser son travail sur un dépôt distant
- Inspecter un dépôt distant
- Retirer et renommer des dépôts distants



Formation Git

alphorm.com™©

## Introduction

- Pour pouvoir collaborer sur un projet Git, il est nécessaire de savoir comment gérer les dépôts distants.
- Les dépôts distants sont des versions de votre projet qui sont hébergées sur Internet ou le réseau d'entreprise.
- Vous pouvez en avoir plusieurs, pour lesquels vous pouvez avoir des droits soit en lecture seule, soit en lecture/écriture.
- Nous allons voir comment :
  - Ajouter des dépôts distants.
  - Effacer des dépôts distants.

## Afficher les dépôts distants

- Pour visualiser les serveurs distants que vous avez enregistré, vous pouvez lancer la commande **'git remote'**.
- Si vous avez cloné notre projet, vous devez avoir au moins un dépôt distant : **'origin'**.
- Vous pouvez aussi spécifier **'-v'**, qui vous montre l'URL que Git a stocké pour chaque nom court.

```
$ git remote -v
bakkdoor https://github.com/bakkdoor/grit (fetch)
bakkdoor https://github.com/bakkdoor/grit (push)
cho45 https://github.com/cho45/grit (fetch)
cho45 https://github.com/cho45/grit (push)
defunkt https://github.com/defunkt/grit (fetch)
defunkt https://github.com/defunkt/grit (push)
koke git://github.com/koke/grit.git (fetch)
koke git://github.com/koke/grit.git (push)
origin git@github.com:mojombo/grit.git (fetch)
origin git@github.com:mojombo/grit.git (push)
```

## Ajouter des dépôts distants

- Pour ajouter des dépôts distants :  
\$> git remote add johnpatrick <https://github.com/johnpatrick/monprojet>

## Récupérer et tirer depuis des dépôts distants

- Pour récupérer les données d'un dépôt distant :  
\$> git fetch [nom du repos distant]
- Cette commande s'adresse au dépôt distant et récupère toutes les données de ce projet que vous ne possédez pas déjà.
- Si vous clonez un dépôt, le dépôt distant est automatiquement ajouté sous le nom « **origin** ».
- Si le dépôt est différent que celui ajouté par défaut, les branches seront ajoutées dans '<nom du dépôt distant>/master'.





## Pousser son travail sur un dépôt distant

- Lorsque votre dépôt vous semble prêt à être partagé, vous pouvez le pousser à votre dépôt distant :

```
$> git push [dépôt distant] [nom-de-la-branche]
```

```
$> git push origin master
```



## Inspecter un dépôt distant

- Pour avoir plus d'informations sur un dépôt distant :

```
$> git remote show [dépôt distant]
```



## Retirer et renommer des dépôts distants

- Pour renommer une référence :  
`$> git remote rename johnpatrick janebob`
- Pour supprimer une référence :  
`$> git remote rm janebob`



## Ce qu'on a couvert

- Nous avons vu dans cette vidéo :
  - Comment afficher, ajouter, renommer et supprimer les dépôts distants
  - Comment pousser et tirer son travail d'un dépôt distant.
  - Comment inspecter un dépôt distant
- La prochaine vidéo :
  - L'Étiquetage





Alphorm.com

## Les bases de Git

### Étiquetage

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>



Édouard FERRARI  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

Formation Git

alphorm.com™©

## Plan

- Introduction
- Lister vos étiquettes
- Créer des étiquettes
- Les étiquettes annotées
- Les étiquettes légères
- Étiqueter après coup
- Partager les étiquettes
- Extraire une étiquette



Formation Git

alphorm.com™©

## Introduction

---

- À l'instar de la plupart des VCS, Git donne la possibilité d'étiqueter un certain état dans l'historique comme important.
- Généralement, on utilise cette fonctionnalité pour marquer les versions d'un projet (v1.0.0, v1.0.42, v2.0.1, ...)
- Nous verrons comment :
  - Lister les différentes étiquettes
  - Comment créer de nouvelles étiquettes
  - Voir tous les types d'étiquettes

## Lister vos étiquettes

---

- Pour lister les étiquettes :

```
$> git tag
```

Les étiquettes sont listées par ordre alphabétique

- Pour faire une recherche :

```
$> git tag -l 'v1.0.*'
```

## Créer des étiquettes

---

- Git utilise deux types d'étiquettes :
  - Étiquette légère
  - Étiquette annotée
- Une **étiquette légère** ressemble beaucoup à une branche qui ne change pas, c'est juste un pointeur sur un commit spécifique.
- Les **étiquettes annotées**, par contre, sont stockées en tant qu'objets à part entière dans la base de données de Git

## Les étiquettes annotées

---

- Créer des étiquettes annotées est simple avec Git :

```
$> git tag -a v1.1.0 -m 'Ma version 1.1.0'
```
- Puis pour afficher en détails votre nouveau tag :

```
$> git show v1.1.0
```

## Les étiquettes légères

---

- Les étiquettes légères vont tout simplement stocker la somme de contrôle d'un commit dans un fichier. Aucune information n'est enregistrée avec l'étiquette.

\$> git tag

## Étiqueter après coup

---

- Il est possible d'étiqueter des anciens commit :

```
git tag -a v1.0.0 9b92f3b
```

## Partager les étiquettes

---

- Par défaut, la commande **git push** ne transfère pas les étiquettes vers les serveurs distants.
- Il faut explicitement pousser les étiquettes après les avoir créées localement.

```
$> git push origin v1.1.0
```

- Pour pouvoir pousser toutes les étiquettes que nous avons créé localement :

```
$> git push origin --tags
```

## Extraire une étiquette

---

- Pour se placer à la position d'un commit étiqueté :  

```
git checkout -b v1.1.0
```
- Grâce à ce mécanisme, nous pouvons naviguer à travers les différentes versions de notre projet

## Ce qu'on a couvert

- Nous avons vu dans cette vidéo toutes les choses que **git** nous propose avec les étiquettes.
- Les étiquettes sont très utilisées en entreprise.
- Prochain chapitre:
  - Les branches avec Git



Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

**Alphorm**.com

## Les branches avec Git

### Les branches en bref



**Édouard FERRARI**  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)



## Plan

---

- Introduction
- Créer une nouvelle branche
- Basculer entre les branches

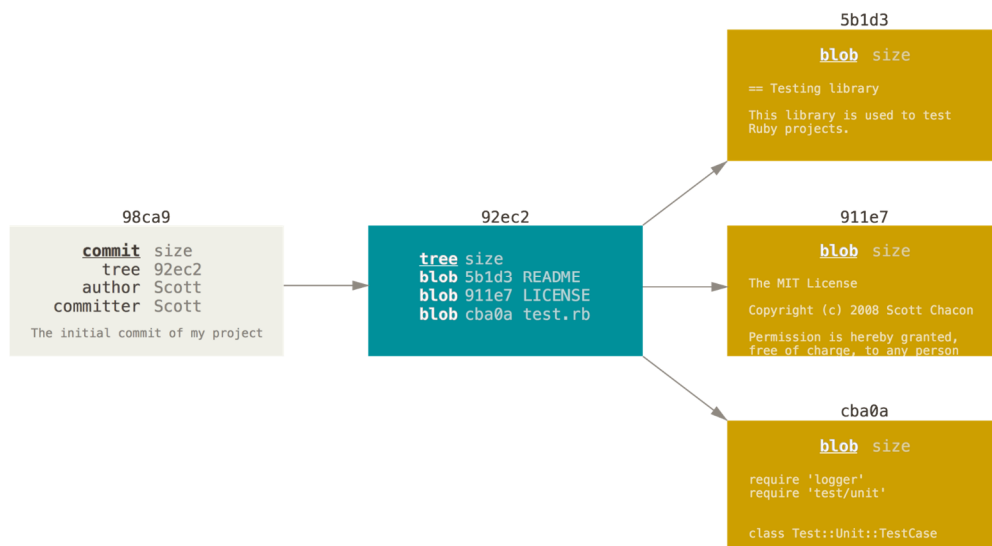


## Introduction

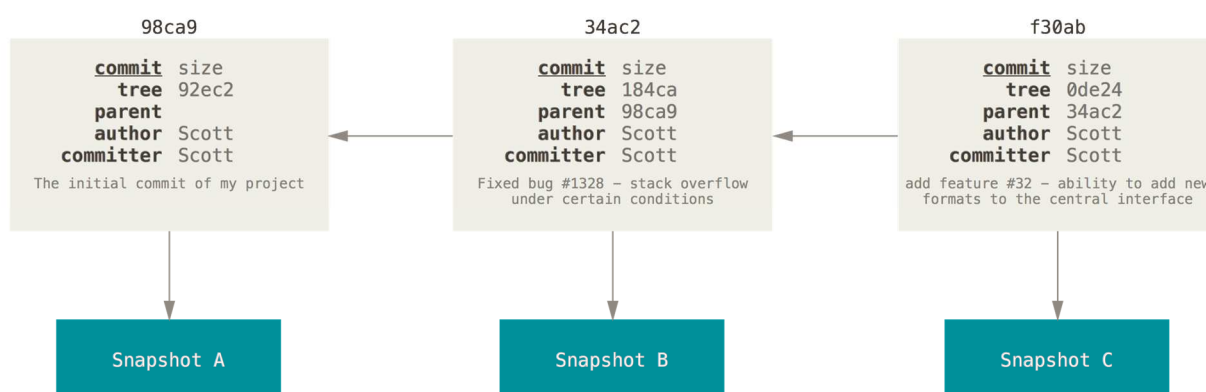
---

- Presque tous les VCS proposent une certaine forme de gestion de branches.
- Créer une branche signifie diverger de la ligne principale de développement et continuer à travailler sans impacter cette ligne.
  - `$> git add README test.rb LICENSE`
  - `$> git commit -m 'initial commit of my project'`

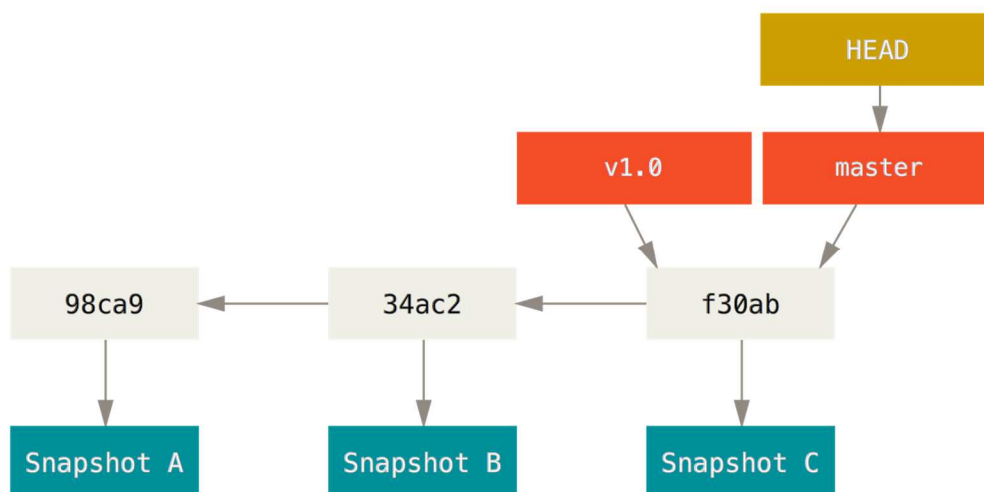
## Introduction



## Introduction



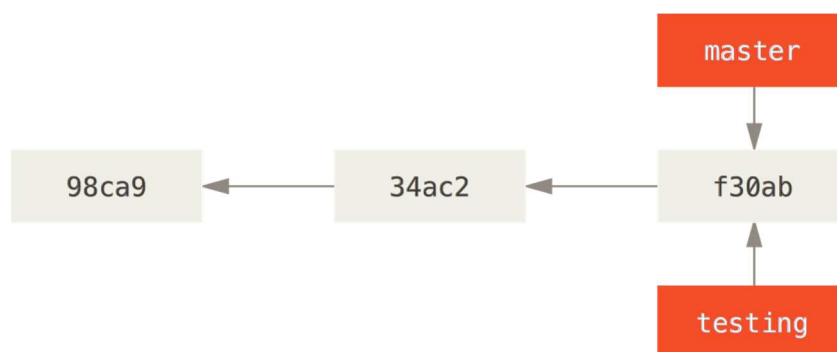
## Introduction



## Créer une nouvelle branche

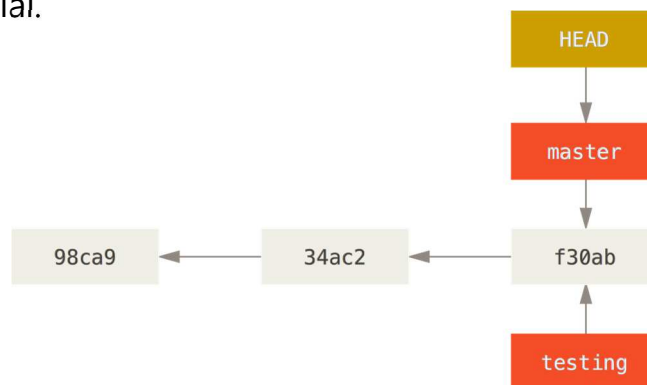
- Pour créer une nouvelle branche :

\$> git branch testing



## Créer une nouvelle branche

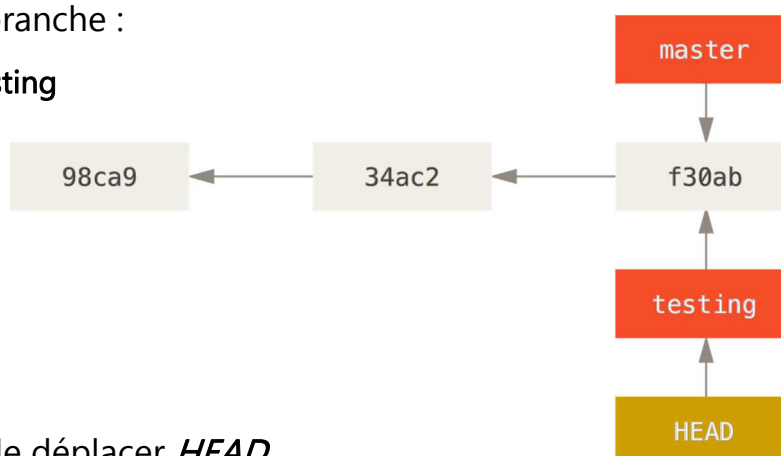
- Pour savoir sur quelle branche **git** se trouve actuellement, il ajoute un pointeur spécial.



- Pour vérifier cela :  
`$> git log --online --decorate`

## Basculer entre les branches

- Pour changer de branche :  
`$> git checkout testing`

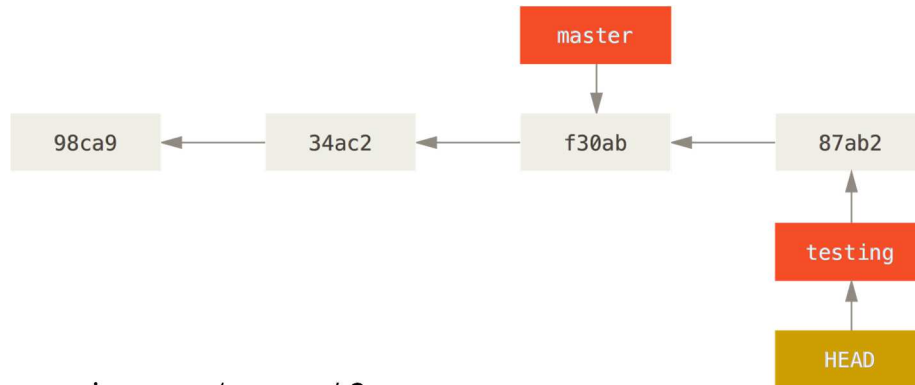


- Cela a pour effet de déplacer *HEAD*.
- Qu'est ce qu'il se passerait si nous créons un commit ?

## Basculer entre les branches

- On édite un fichier et on crée un commit :

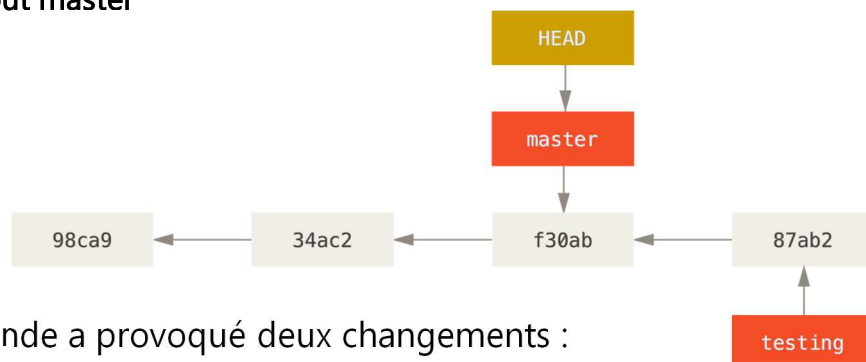
\$> git commit -a -m 'nouvelle modification'



- Et si on revient sur '*master*' ?

## Basculer entre les branches

\$> git checkout master

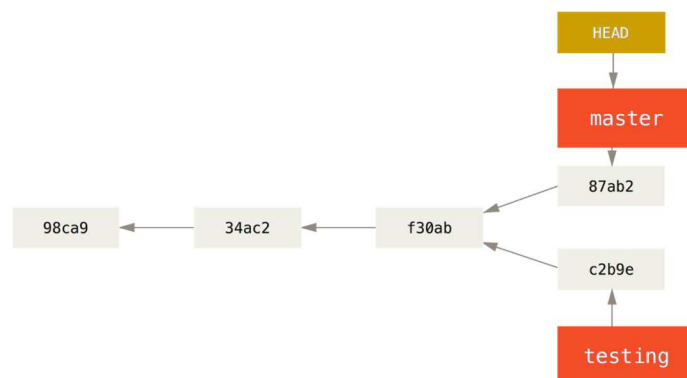


- Cette commande a provoqué deux changements :
  - Mettre le pointeur *HEAD* sur *master*.
  - Elle a remplacé les fichiers de votre répertoire dans l'état du snapshot pointé par *master*.

## Basculer entre les branches

- On continue nos tests.
- Faisons encore une modification dans un fichier et créons un nouveau commit.
- À quoi ressemble notre arbre ?

## Basculer entre les branches



- On peut retrouver cet arbre avec :  
\$> git log --oneline --decorate --graph --all  
\$> gitk --all

## Ce qu'on a couvert

---

- Nous avons vu et compris :
  - Comment fonctionnent les branches
  - Comment les créer
  - Comment sauter d'une branche à l'autre.
- La prochaine vidéo :
  - Branches et fusions : les bases



Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

**Alphorm**.com

## Les branches avec Git

---

### Branches et fusions : les bases



**Édouard FERRARI**  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

## Plan

---

- Scénario
- Mise en place
- Développement du correctif
- Téléphone
- Modification du site internet
- Retour sur le correctif
- Une fusion un peu spéciale



## Scénario

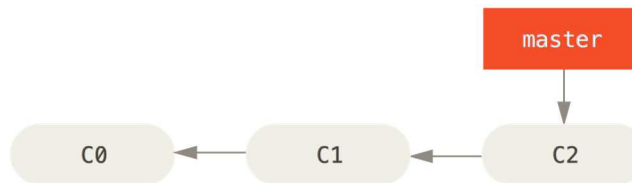
---

- Vous effectuez les tâches suivantes :
  - vous travaillez sur un site web ;
  - vous créez une branche pour un nouvel article en cours ;
  - vous commencez à travailler sur cette branche.
- À cette étape, vous recevez un appel pour vous dire qu'un problème critique a été découvert et qu'il faut le régler au plus tôt. Vous faites donc ce qui suit :
  - vous basculez sur la branche de production ;
  - vous créez une branche pour y ajouter le correctif ;
  - après l'avoir testé, vous fusionnez la branche du correctif et poussez le résultat en production ;
- vous rebasculez sur la branche initiale et continuez votre travail.



## Mise en place

- Nous avons notre projet qui ressemble à ça :



- On décide de travailler sur le problème #53, on crée une branche et on se place dessus.

\$> git checkout -b iss53

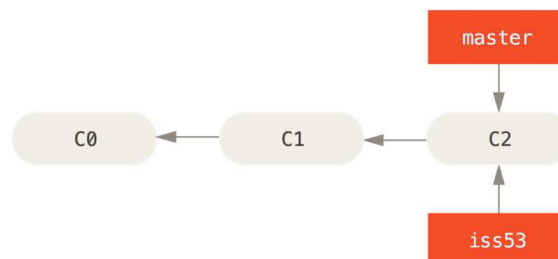
- Équivalent à :

\$> git branch iss53

\$> git checkout iss53

## Développement du correctif

- Notre projet ressemble donc à cela et *HEAD* est sur iss53.

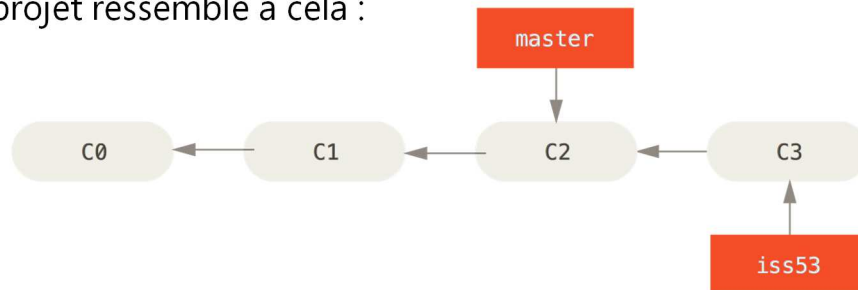


- Nous faisons une modification et nous créons un nouveau commit :

\$> git commit -a -m 'Résolution d\'un premier bug'

## Téléphone

- Notre projet ressemble à cela :



- Le téléphone sonne, il faut qu'on fasse immédiatement une modification sur le site internet.

\$> git checkout master

## Modification du site internet

- Nous allons développer notre modification urgente en créant une nouvelle branche:

\$> git checkout -b hotfix

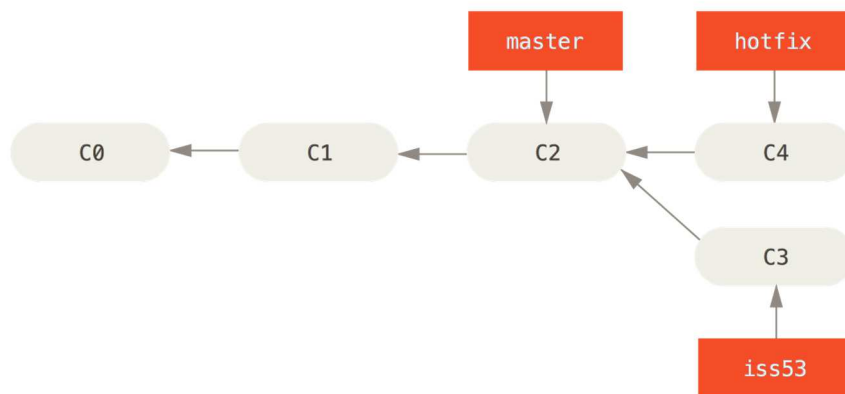
- On modifie notre code.
- On crée un commit :

\$> git commit -a -m 'correction de l'adresse email'

- À quoi ressemble notre arbre ?

## Modification du site internet

- Notre arbre ressemble à :



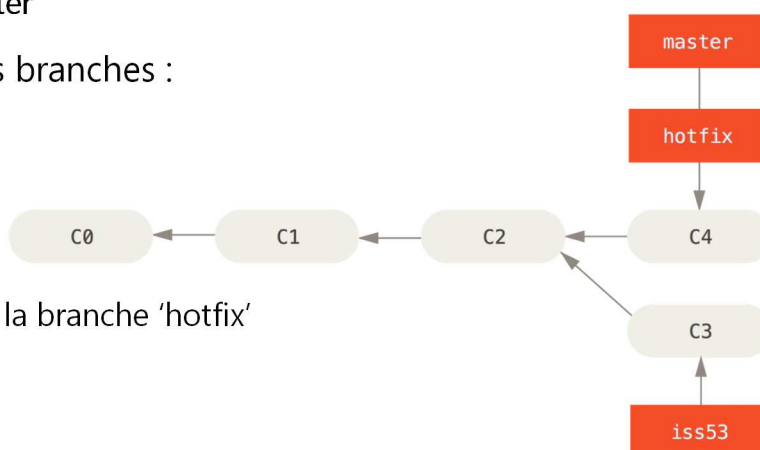
## La fusion

- Maintenant, nous revenons sur master :

\$> git checkout master

- Nous fusionnons les branches :

\$> git merge hotfix

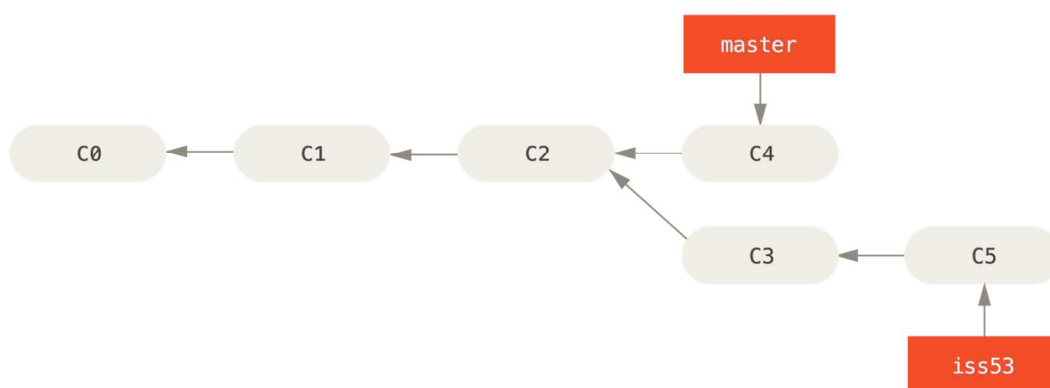


- On peut supprimer la branche 'hotfix'

## Retour sur le correctif

- On retourne sur le correctif  
`$> git checkout iss53`
- On continue nos développements
- On commit les nouvelles modifications  
`$> git commit -a -m 'Résolution deuxième bug, donc problème #53 corrigé'`
- À quoi ressemble notre arbre ?

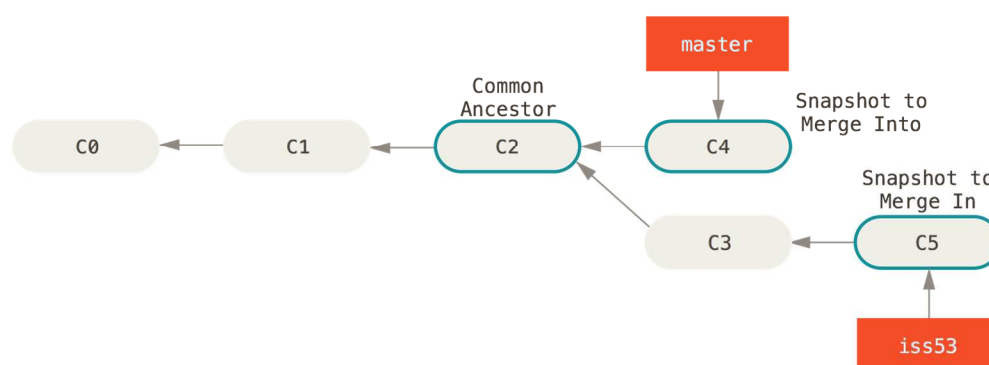
## Retour sur le correctif



## Une fusion un peu spéciale

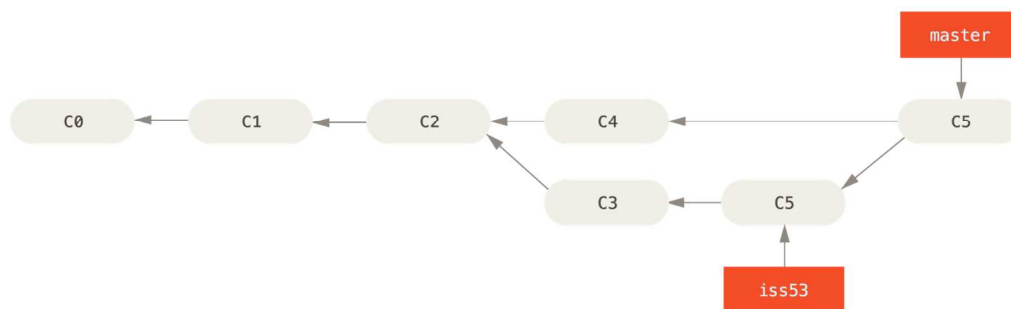
- On se place sur 'master' :  
\$> git checkout master
- On fusionne les deux branches  
\$> git merge iss53
- Qu'est ce qu'il se passe dans ce cas ?

## Une fusion un peu spéciale



## Une fusion un peu spéciale

- Git crée un nouveau commit qui contient le fusion de C4 et C5.



## Un conflit

- Si nous modifions le même fichier dans deux branches qui seront fusionnées, il est possible que Git n'arrive plus à faire une fusion.
- La procédure est la suivante :
  - \$> git checkout master
  - \$> git merge future\_conflit
- Un conflit apparait, on rentre dans le(s) fichier(s) incriminé(s) et corrige le conflit.
  - \$> git add <les fichiers>
  - \$> git commit -m 'message'

## Ce qu'on a couvert

- Nous avons un scénario typique d'utilisation des branches en entreprise.
- Prochaine vidéo:
  - Gestion des branches



Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

**Alphorm**.com

## Les branches avec Git

### Gestion des branches



**Édouard FERRARI**  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

## Plan

---

- Lister les branches
- Les branches fusionnées et non fusionnées
- Supprimer une branche



## Lister les branches

---

- Maintenant que vous avez créé, fusionné et supprimé des branches, regardons de plus près les outils.
- 'git branch', sans argument, permet de lister les branches en local.  
\$> git branch
- Pour avoir un peu plus d'informations  
\$> git branch -v





## Les branches fusionnées et non fusionnées

- Les options --merged et --no-merge sont des options très utiles.
- Pour avoir la liste des branches déjà fusionnées:  
`$> git branch --merged`
- Pour avoir la liste des branches pas encore fusionnées :  
`$> git branch --no-merged`



## Supprimer une branche

- Pour supprimer une branche :  
`$> git branch -d <branche à supprimer>`

## Ce qu'on a couvert

- Dans cette vidéo, nous avons vu comment lister les branches
  - Toutes les branches locales.
  - Toutes les branches déjà fusionnées.
  - Toutes les branches pas encore fusionnées.
- La prochaine vidéo :
  - Travailler avec les branches



Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

**Alphorm**.com

## Les branches avec Git

### Travailler avec les branches



**Édouard FERRARI**  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

## Plan

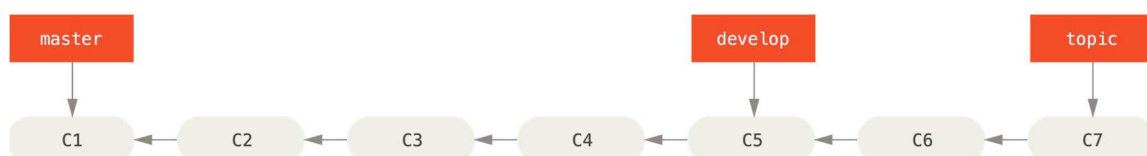
---

- Branche au long cours
- Branche thématique

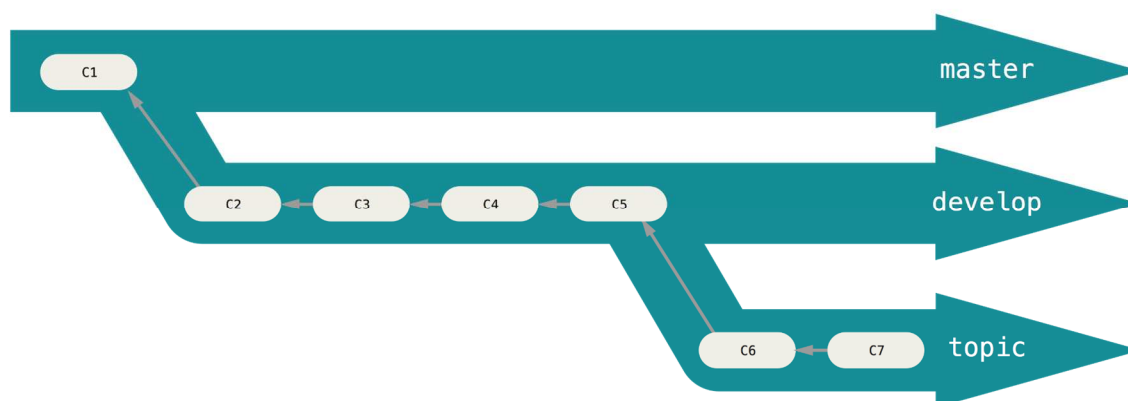


## Branche au long cours

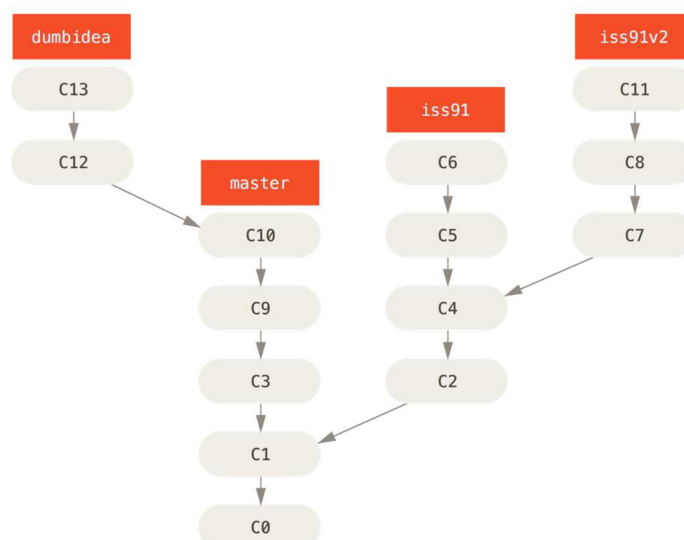
---



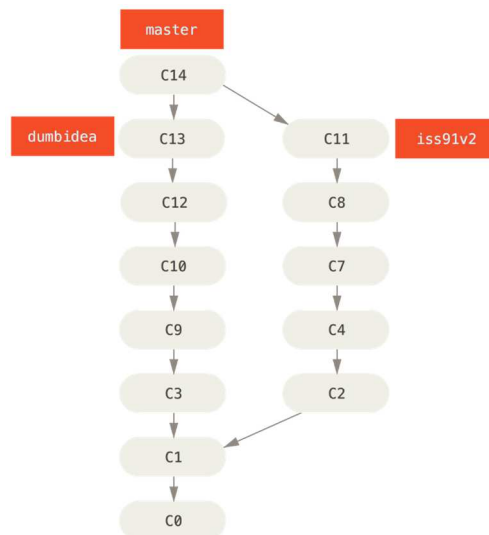
## Branche au long cours



## Branche thématique



## Branche thématique



## Ce qu'on a couvert

- Nous avons vu dans cette vidéo :
  - Différent type de branches.
  - Comment on s'organise en entreprise.
- Vidéo suivante :
  - Les branches distantes





Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

Formation Git

**Alphorm**.com

## Les branches avec Git

### Introduction aux branches distantes



Édouard FERRARI  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

alphorm.com™©

## Plan

- Branches distantes
- Pousser les branches



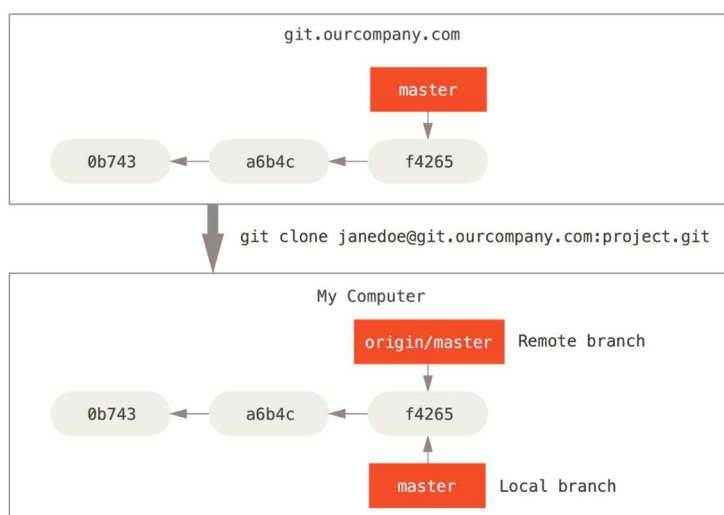
Formation Git

alphorm.com™©

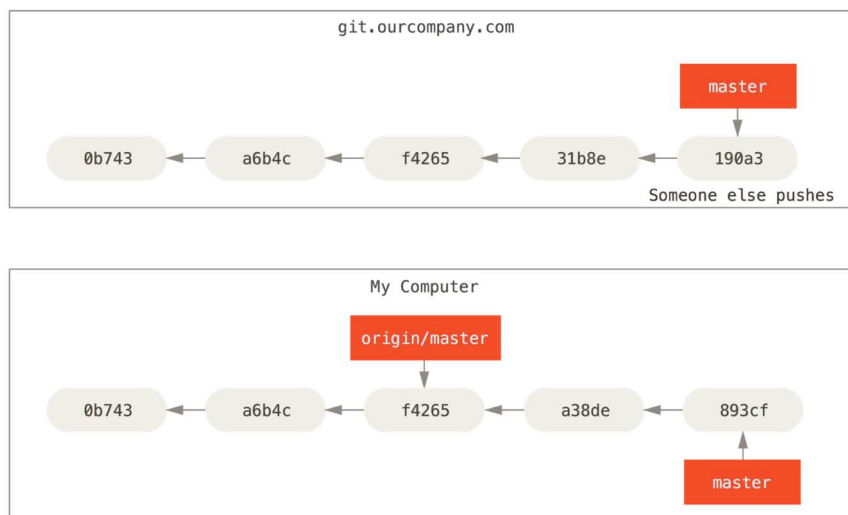
## Branches distantes

- Les références distantes sont des références (pointeurs) vers les éléments de votre dépôt distant tels que les branches, les tags, etc...
- Pour obtenir la liste complète de ces références :  
`$> git ls-remote <remote>`
- Les branches distantes sont des références (des pointeurs) vers l'état des branches sur votre dépôt distant.
- Elles sont sous la forme <distant>/<branche>

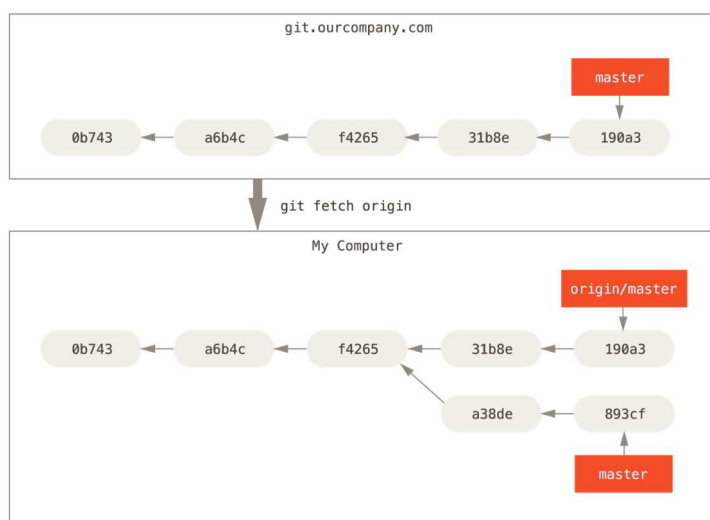
## Branches distantes



## Branches distantes

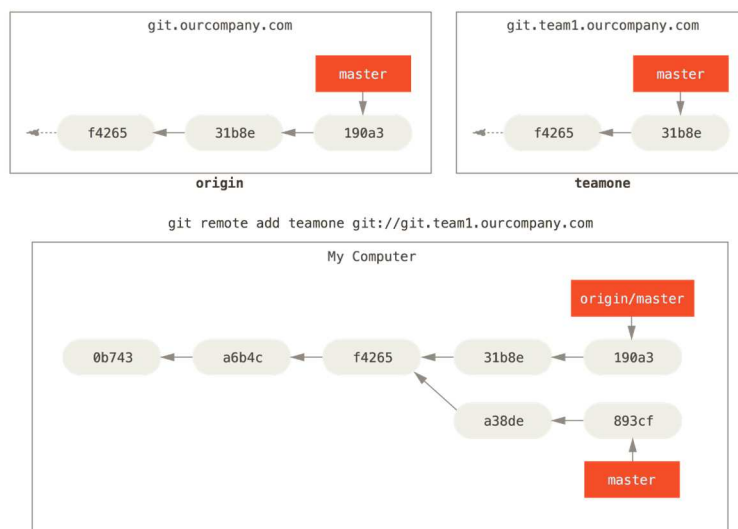


## Branches distantes

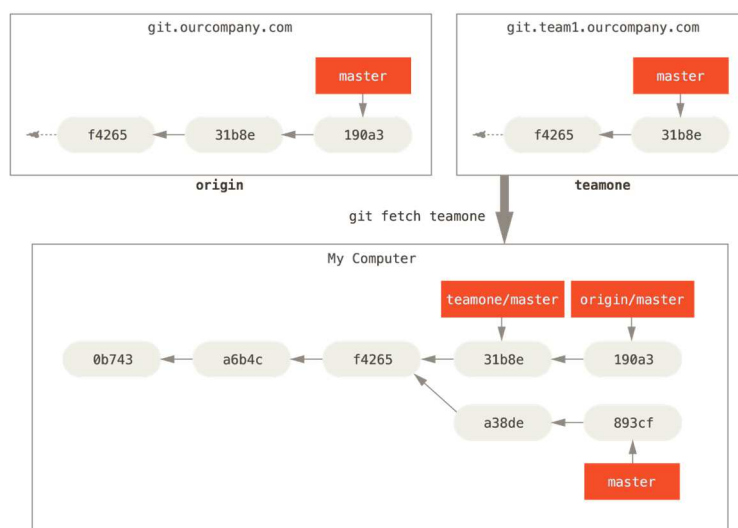




## Branches distantes



## Branches distantes



## Pousser les branches

---

- Vos branches locales ne sont pas automatiquement synchronisées sur les serveurs distants.
- Vous devez pousser explicitement les branches que vous souhaitez partager.
- Pour pusher une branche :  

```
$> git push <distant> <branche>
```
- La prochaine fois qu'un de vos collègues récupérera les données depuis le serveur, il récupérera la branche distante créée.

## Pousser les branches

---

- Il est important de noter que lorsque vous récupérez une nouvelle branche depuis un serveur distant, vous ne créez pas automatiquement une copie locale éditale.
- Mais seulement un pointeur sur la branche distante qui n'est pas directement modifiable.
- Pour fusionner ce travail dans votre branche de travail :  

```
$> git merge origin/branchejohnpatrick
```
- Si vous ne souhaitez pas fusionner la branche distante avec votre travail, vous pouvez créer une nouvelle branche et copier le travail dedans :  

```
$> git checkout -b branchejohnpatrick origin/branchejohnpatrick
```

## Ce qu'on a couvert

---

- Nous avons vu dans cette vidéo :
  - Comment s'organiser en entreprise.
  - Comment pousser une branche distante.
- Vidéo suivante :
  - Interagir avec les branches distantes



Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

# Alphorm.com

## Les branches avec Git

---

### Interagir avec les branches distantes



**Édouard FERRARI**  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

## Plan

---

- Suivre les branches
- Tirer une branche (Pulling)
- Suppression de branches distantes



## Suivre les branches

---

- L'extraction d'une branche locale à partir d'une branche distante crée automatiquement ce qu'on appelle une « **branche de suivi** ».
- Si nous rentrons la commande '**git push**', Git sélectionne automatiquement le serveur vers lequel pousser vos modifications.
- Pour connaître la branche suivie :  

```
$> git branch -vv
```
- Pour changer la branche suivie :  

```
$> git branch -u origin/nouvellebranchesuivie
```

```
$> git branch --set-upstream-to origin/nouvellebranchesuivie
```

## Tirer une branche (Pulling)

- La commande '**git fetch**' récupère l'ensemble des changements présents sur le serveur, mais elle ne modifie en rien votre répertoire de travail.
- Il faut faire un '**git merge**' pour fusionner les nouveaux changements.
- La commande '**git pull**' va faire un '**git fetch**' et '**git merge**' d'un coup.

\$> git pull

## Suppression de branches distantes

- Pour supprimer une branche :
- \$> git push origin --delete <branche>

## Ce qu'on a couvert

---

- Nous avons vu dans cette vidéo :
  - Comment suivre une branche distante.
  - Comment tirer une branche.
  - Comment supprimer une branche.
- Vidéo suivante :
  - Rebaser (Rebasing)



Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

**Alphorm**.com

## Les branches avec Git

---

### Rebaser (Rebasing)



**Édouard FERRARI**  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

## Plan

---

- Introduction
- Les bases
- Rebases plus intéressant
- Les dangers du rebase
- Rebaser quand vous rebasez
- Rebaser ou fusionner ?

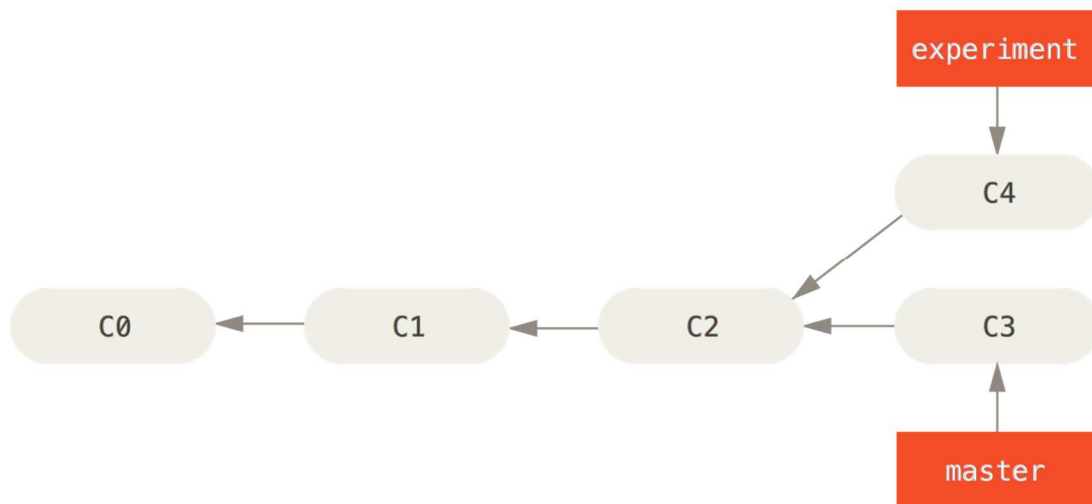


## Introduction

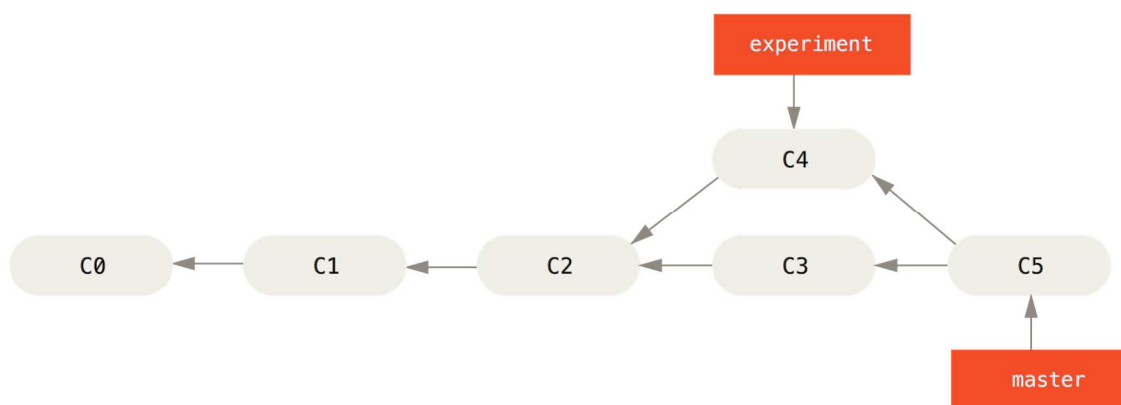
---

- Il y a deux façons d'intégrer des modifications dans une branche :
  - En fusionnant (*merge*)
  - En rebasant (*rebase*)

## Les bases



## Les bases

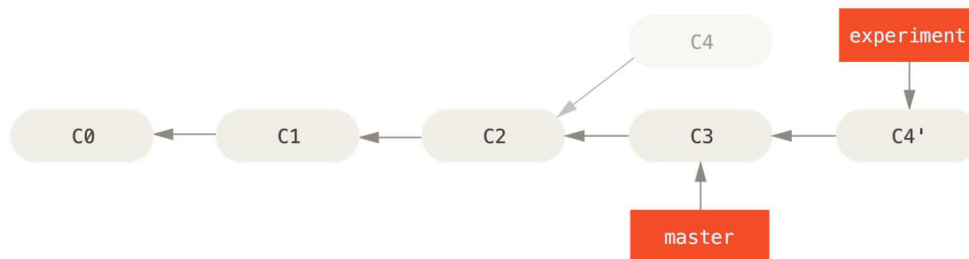




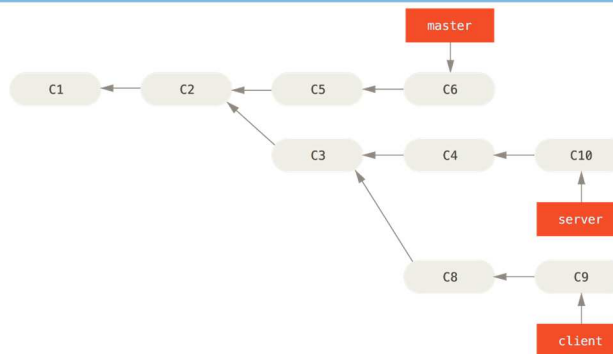
## Les bases

- La commande '**rebase**', Git va prendre toutes les modifications qui ont été validées sur une branche et les rejouer sur une autre.

\$> git rebase master



## Rebases plus intéressant



- On veut fusionner nos modifications de '**client**' avec '**master**' mais on veut retenir la branche '**master**'

\$> git rebase --onto master server client

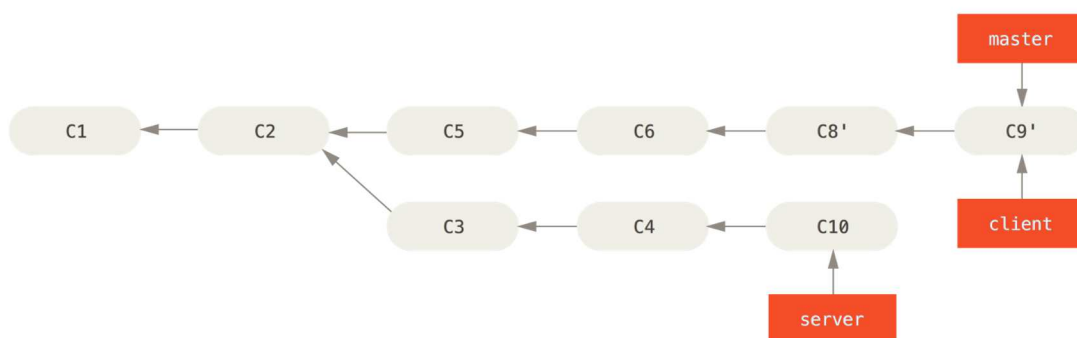
- "Extraire la branche client, déterminer les patches depuis l'ancêtre commun des branches client et serveur puis les rejouer sur master "

## Rebases plus intéressant

- Et maintenant, on peut avancer 'master'.

\$> git checkout master

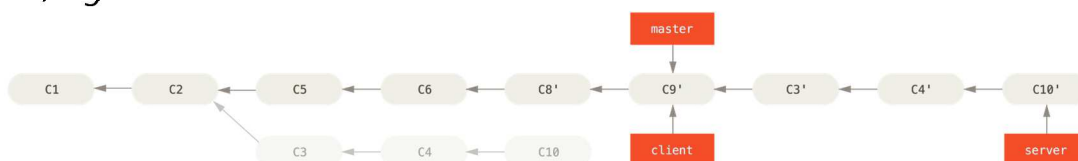
\$> git merge client



## Rebases plus intéressant

- Et maintenant, la branche 'server' à partir de la branche 'master'

\$> git rebase master server



- Une avance rapide sur la branche 'master':

\$> git checkout master

\$> git merge serveur

## Rebases plus intéressant

- Et on peut supprimer les branches restantes :

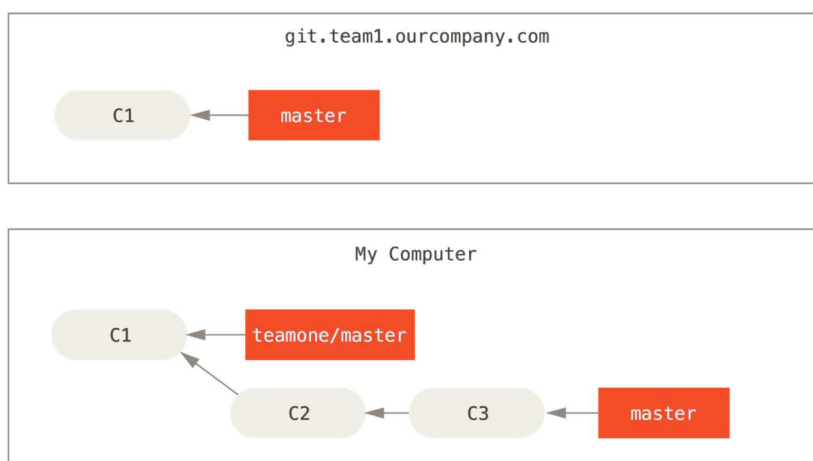
\$> git branch -d client

\$> git branch -d server

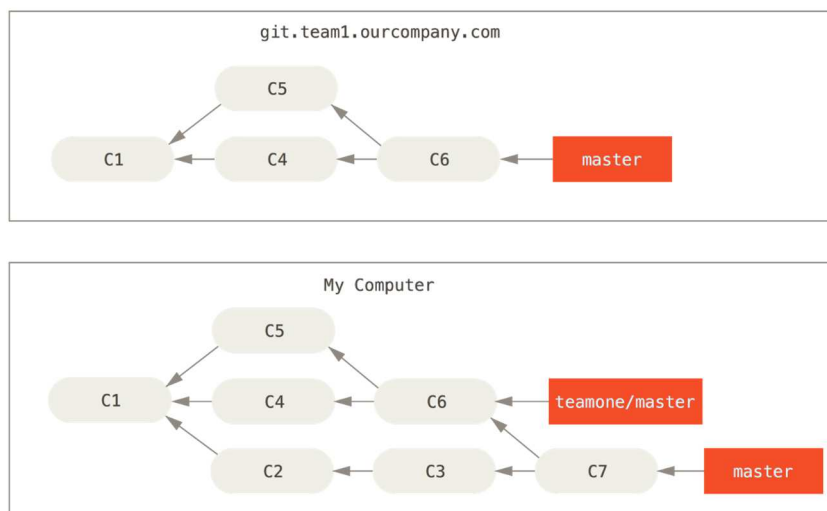


## Les dangers du rebasage

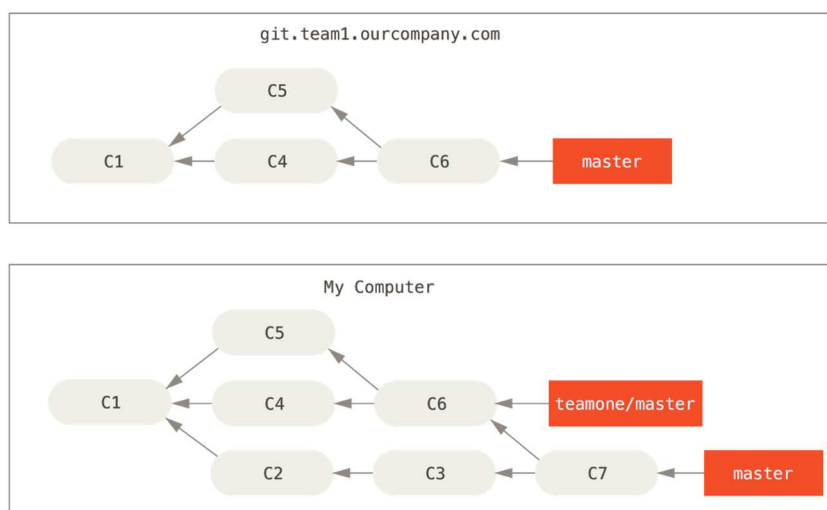
- **Ne rebasez jamais** des commits qui ont déjà été poussés sur un dépôt public.



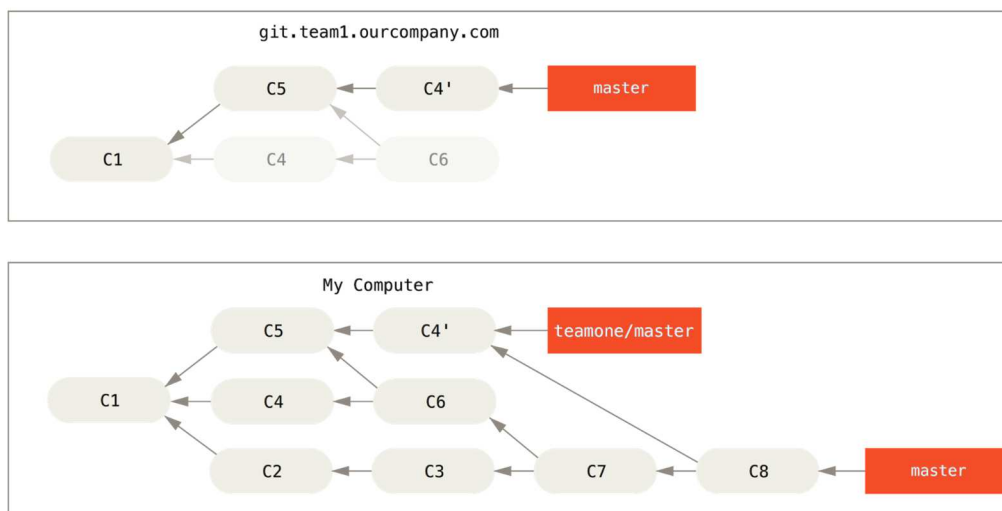
## Les dangers du rebasage



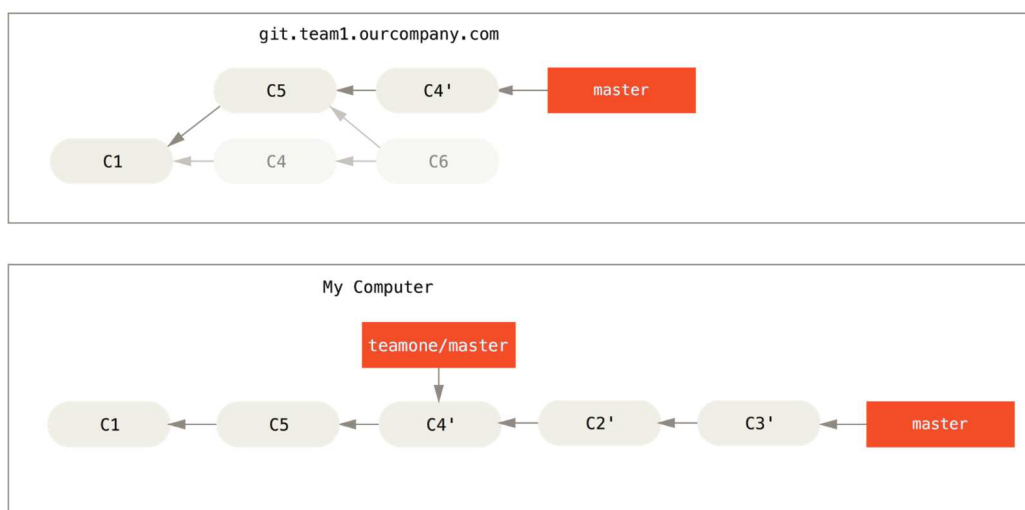
## Les dangers du rebasage



## Les dangers du rebasage



## Rebaser quand vous rebasez





## Rebaser ou fusionner

---

- C'est très important de conserver votre historique intact.
- Une fusion (merge) va créer un commit
- Un rebase va réagencer les commits et suivre l'historique



## Ce qu'on a couvert

---

- Nous maîtrisons le rebase parfaitement.
- Nous avons vu plusieurs exemples de rebase.
- À vous de faire un choix ! ;-)
  
- Prochain chapitre:
  - Github





**Alphorm**.com

## Github

### Création et configuration d'un compte

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>



**Édouard FERRARI**  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

Formation Git

alphorm.com™©

## Plan

- Inscription
- Accès par SSH
- Vos adresses emails
- Authentification à deux facteurs



Formation Git

alphorm.com™©



## Inscription

---

- Créons-nous un compte sur : <https://github.com>
- N'oubliez pas de vérifier votre compte grâce à l'email que vous avez reçu.



## Accès par SSH

---

- Si vous voulez joindre les serveurs de github par SSH, il faut ajouter vos clefs SSH.
- Setting -> SSH keys
- Et vous ajoutez votre clef publique.



## Vos adresses électroniques

---

- Github utilise les adresses électroniques pour faire correspondre les commits Git aux utilisateurs.
- Pour ajouter les adresses électroniques :
  - Setting -> Emails

## Authentification en deux facteurs

---

- L'authentification à deux facteurs est un mécanisme d'authentification qui est devenu très populaire récemment pour réduire les risques de corruption de votre compte si votre mot de passe est dérobé.
  - Setting -> Security
- Soit par une application
- Soit par un SMS

## Ce qu'on a couvert

---

- Nous avons désormais un compte git
- La prochaine vidéo :
  - Contribution à un compte



Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

**Alphorm**.com

Github

---

Contribution  
à un projet



**Édouard FERRARI**  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

## Plan

---

- Dupliquer un projet
- Processus Github



## Dupliquer un projet

---

- C'est simple !
  - <https://github.com/didouard/project-alphorm>



## Processus Github

---

- Le principe général est le suivant :
  1. Création d'une branche thématique à partir de la branche master
  2. Validation de quelques améliorations (commit)
  3. Poussée de la branche thématique sur votre projet GitHub (push)
  4. Ouverture d'une requête de tirage sur GitHub (Pull Request)
  5. Discussion et éventuellement possibilité de nouvelles validations (commit)
  6. Le propriétaire du projet fusionne (merge) ou ferme (close) la requête de tirage

## Ce qu'on a couvert

---

- Nous avons vu comment collaborer sur github.
- Prochaine vidéo :
  - Github flavored Markdown





Alphorm.com

Github

## Github flavored Markdown

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>



Édouard FERRARI  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

Formation Git

alphorm.com™©

### Plan

- Introduction
- Les différents markdown



Formation Git

alphorm.com™©



## Introduction

---

- Il est possible d'enrichir votre texte dans presque toutes les boîtes de saisies :
  - Descriptions d'anomalies
  - Requêtes de tirage
  - Les commentaires
  - Les commentaires de code
  - ...



## Les différents markdown

---

- Liste de tâches
- Extrait de code
- Citation
- Émoticône
- Images

## Ce qu'on a couvert

---

- Nous avons vu comment enrichir nos requêtes, nos commentaires ...
- Prochaine vidéo :
  - Maintenance d'un projet



**Alphorm**.com

Github

---

Maintenance  
d'un projet

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>



**Édouard FERRARI**  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

## Plan

---

- Création d'un nouveau dépôt
- Ajout de collaborateurs
- Gestion des requêtes de tirage
- Mentions et notifications
- Fichiers spéciaux



## Création d'un nouveau dépôt

---

- Créons un nouveau dépôt.
  - Name : **myproject-alphorm**
- Notre projet est désormais clonable à partir de :  
`$> git clone https://github.com/<utilisateur>/<nom\_du\_projet>`
- Si votre projet existe déjà, nous pouvons ajouter un lien distant  
`$> git remote add origin https://github.com/<utilisateur>/<nom\_du\_projet>`



## Ajout de collaborateurs

- Il est possible d'ajouter des collaborateurs en accès poussé.
- Il faut que chaque collaborateur ai un compte github.
- Une fois les comptes ajoutés, les collaborateurs auront les droits en lecture et en écriture.
  - Setting (Du projet) -> Collaborators

## Gestion des requêtes de tirage

- Si quelqu'un vous demande un pull request, vous recevrez un email.

[didouard/example-formation] 3000 is best (#1)

[VIEW PULL REQUEST](#)



patrick-alphorm à didouard/example-formation ↗

Salut, 3000, c'est mieux

You can view, comment on, or merge this pull request online at:

<https://github.com/didouard/example-formation/pull/1>

Commit Summary

- 3000 is best

File Changes

- M [bin/www](#) (2)

Patch Links:

- [https://github.com/didouard/example-formation/pull/1\\_patch](https://github.com/didouard/example-formation/pull/1_patch)
- [https://github.com/didouard/example-formation/pull/1\\_diff](https://github.com/didouard/example-formation/pull/1_diff)

You are receiving this because you are subscribed to this thread.  
Reply to this email directly or [view it on GitHub](#)

## Gestion des requêtes de tirage

---

- À partir de là, on peut lancer une conversation si nous avons des questions concernant ce pull request.
- Une fois que vous êtes ok pour le pull request, vous avez le choix
  - De faire un '`git pull <url> <branch>`'
  - D'ajouter le clone comme un dépôt distance, le récupérer et le fusionner
  - De cliquer sur le bouton '**Merge**' sur le site internet.

## Mentions et notifications

---

- GitHub dispose également d'un système de notifications qui peut devenir utile lorsque vous avez des questions de certaines personnes ou d'équipes.
- Dans tous les commentaires, si vous saisissez le caractère @, cela commence à proposer des noms et des noms d'utilisateur de personnes qui collaborent ou contribuent au projet.
- Une fois le commentaire posté, tous ceux qui sont tagués recevront une notification.
- Il est possible de se désinscrire de ces notifications :
  - Setting -> Notification center

## Fichiers spéciaux

---

- Le premier est le fichier **README** (LISEZ-MOI) qui peut être écrit sous n'importe quel format textuel reconnu par GitHub.
- Il contient habituellement des choses comme :
  - À quoi sert le projet.
  - Comment le configurer et l'installer.
  - Un exemple d'utilisation et comment le lancer.
  - La licence sous laquelle le projet est proposé.
  - Comment y contribuer.

## Ce qu'on a couvert

---

- Nous avons vu comment gérer un projet.
- Prochaine vidéo :
  - Gestion d'un regroupement





Alphorm.com

Github

## Gestion d'un regroupement

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>



Édouard FERRARI  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

Formation Git

alphorm.com™©

### Plan

- Introduction
- Les bases d'un regroupement
- Équipes
- Journal d'audit



Formation Git

alphorm.com™©

## Introduction

---

- Les regroupements sont des 'Organisations'.
- Ce sont comme des comptes personnels :
  - Dépôts
  - Suivi
- Ces comptes représentent un groupe de personnes qui partagent la propriété de projets et de nombreux outils de gestion de sous-groupes parmi ces personnes sont proposés.
- Normalement ces comptes sont utilisés pour des groupes open source (tels que « perl » ou « rail ») ou des sociétés (comme « google » ou « twitter »).

## Les bases d'un regroupement

---

- Pour créer un regroupement :



- Puis il faut trouver :
  - Un nom
  - Un email
  - Inviter d'autres utilisateurs
  - Les regroupements sont gratuits si tout ce que vous envisagez d'enregistrer est open source.

## Équipes

---

- Il est possible de créer des équipes (team)
- On peut associer certains utilisateurs à certaines équipes
- Lorsque vous invitez quelqu'un dans une équipe, celui-ci reçoit un courriel lui indiquant qu'il a été invité.

## Ce qu'on a couvert

---

- Nous avons vu les regroupements pour les grandes entreprises ou les grands projets.
- Prochaine vidéo:
  - Écriture de scripts pour GitHub





Alphorm.com

## Github

# Écriture de scripts pour GitHub (Hooks)

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>



Édouard FERRARI  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

Formation Git

alphorm.com™©

## Plan

- Les services
- Les webhooks
- L'interface de programmation (API) GitHub
- Commenter un problème
- Changer le statut d'une requête de tirage
- Octokit



Formation Git

alphorm.com™©

## Les services

---

- La section « **Hooks & Services** » (crochets et services) de l'administration de dépôt GitHub est la façon la plus facile de faire interagir GitHub avec des systèmes externes.
- Ils sont accessibles dans **Setting -> Webhooks et Services**
- On trouve énormément de liens vers :
  - des services d'intégration continue
  - des analyseurs de bogues et d'anomalies
  - des systèmes de salon de discussion
  - des systèmes de documentation

## Les webhooks

---

- Si vous avez besoin de quelque chose de plus spécifique, quelque chose qui n'est pas présent dans les services, on peut utiliser les webhooks.
- Les webhooks sont des inscriptions à des événements.
- À chaque événement, github va appeler une URL en POST



## L'interface de programmation (API) GitHub

- Github met à disposition une API pour avoir plus d'informations.
- <https://developer.github.com/v3/>
- Presque tout ce que nous pouvons faire à partir du site internet de Github peut être fait avec l'API
- Certaines requêtes nécessitent une authentification.
- Exemple simple :

```
$> curl https://api.github.com/users/<username>
```

```
$> curl https://api.github.com/gitignore/templates/Java
```

## L'interface de programmation (API) GitHub

- Il y a plusieurs méthodes d'authentification :
  - Basic
  - Jeton d'accès personnel (Personnal setting -> Personal access token)
- Exemple plus complexe :

```
$> curl https://api.github.com/repos/<utilisateur>/<dépôt>/issues/<num>/comments
```

  - ```
curl -H "Content-Type: application/json" \  
  -H "Authorization: token TOKEN" \  
  --data '{"body":"A new comment, :+1:"}' \  
  https://api.github.com/repos/didouard/blink/issues/6/comments
```



## L'interface de programmation (API) GitHub

- On peut faire un peu près tout ce que l'on peut faire avec le site internet :
  - créer et définir des jalons
  - assigner des gens à des problèmes ou à des requêtes de tirage
  - créer et changer des étiquettes
  - accéder à des données de *commit*
  - créer de nouveaux commits et des branches
  - ouvrir, fermer ou fusionner des requêtes de tirage,
  - créer et éditer des équipes
  - commenter des lignes de code dans une requête de tirage
  - bien plus encore.



## Octokit

- Octokit est le nom de plusieurs clients pour l'API Github, aujourd'hui disponible :
  - .NET
  - Ruby rookit
  - Simple Go
  - Objective-C
  - Python
- <https://github.com/octokit>

## Ce qu'on a couvert

- Nous avons vu comment interagir avec Github, soit
  - En s'inscrivant à des événements.
  - En interrogeant ou modifiant nos dépôts sur Github.
- Prochaine vidéo:
  - La conclusion !



**Alphorm**.com

## Conclusion

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>



**Édouard FERRARI**  
Formateur et Consultant indépendant  
Contact : [edouard.ferrari@gmail.com](mailto:edouard.ferrari@gmail.com)

## Ce qu'on a couvert

---

- Comment installer et paramétrer GIT
  - Les différents états des fichiers
  - Les branches, les fusions et les rebases
  - Les branches distantes
  - Création de comptes github
  - Contribution et gestion d'un projet Github
- 
- Prochaine formation:
    - Git, avancé

## Avez-vous des Questions / Remarques / Commentaires ?

---



## À la 2<sup>ème</sup> formation sur Git

---



Formation Git

alphorm.com™©

## À bientôt ☺

---

### Keep in touch !

E-mail : [edouard@ferrari.wf](mailto:edouard@ferrari.wf)  
Linkedin : <https://fr.linkedin.com/in/edouardferrari>  
Twitter : [https://twitter.com/edouard\\_ferrari](https://twitter.com/edouard_ferrari)  
Alphorm : <http://www.alphorm.com/formateur/edouard-ferrari>



Formation Git

alphorm.com™©