

GIT

# Sommaire

# Sommaire

- ❑ Introduction
- ❑ Installation et configuration
- ❑ Création et initialisation d'un dépôt local
- ❑ Création d'un dépôt Github et clônage
- ❑ Modification d'un dépôt
- ❑ Historique d'un dépôt
- ❑ Gestion des branches
- ❑ GitFlow

# Introduction

# Introduction

## ■ Git

### ○ Présentation

- Outils de versionning créé en 2005 par Linus Torvalds, le créateur de Linux
- Permet de conserver un historique des modifications effectuées sur un projet :
  - Identifier les changements effectués
  - Revenir à une ancienne version en cas de problème.
- Facilite grandement la gestion de projets et car ils permettent de travailler en équipe de manière beaucoup plus efficace.

# Introduction

## □ Git

### ○ Dépôt

- Copie de l'ensemble des fichiers d'un projet dans Git.

Deux façons de créer un dépôt Git :

- Importer un répertoire existant dans Git ;
- Cloner un dépôt Git déjà existant.

### ○ Flux d'instantanés ou “snapshots”

- Enregistrement instantané du contenu du projet à un instant T avec une référence pour qu'on puisse y accéder par la suite.
- Chaque instantané est stocké dans une base de donnée locale.

# Introduction

## ■ Git

### ○ État des fichiers

#### ○ Deux grands états :

- sous suivi de version : appartient au dernier instantané.
  - Modifié (“modified”) ;
  - Indexé (“staged”) ;
  - Validé (“committed”).
- non suivi (“unstaged”) : tout fichier qui n’appartenait pas au dernier instantané et qui n’a pas été indexé.

#### ○ Fichier indexé :

- Fichier ajouté, modifié ou supprimé devant faire partie du prochain instantané.

#### ○ Fichier validé :

- Fichier enregistré dans l’instantané.

# Introduction

## ■ Git

### ○ État des fichiers

#### ○ Création d'un projet :

- Tous les fichiers sont non suivis. Il faut demander à Git de les indexer et de les valider (les enregistrer dans la base de données).

#### ○ Clonage d'un projet :

- Tous les fichiers sont déjà suivis car on récupère entièrement l'historique du projet

#### ○ Modification d'un projet :

- Nouveaux fichiers considérés comme « unstaged »
- Fichiers modifiés considérés comme « modified » tant qu'ils ne sont pas indexés
- Les fichiers modifiés ou nouveaux doivent être indexés pour être ensuite validés.



# Introduction

## ■ Git

### ○ Les zones de travail

#### ○ Liées aux états de fichiers

#### ○ Trois sections par projet Git :

- Répertoire de travail (working tree) : Répertoire des fichiers du projet informatique.
- Zone d'index (staging area) : Fichier stockant les informations pour le prochain instantané.
- Répertoire Git (repository) : Méta-données et base de données des objets du projet informatique.

#### ○ Le processus de travail :

- Développement informatique dans le répertoire de travail
- Lorsque l'on indexe un fichier, son nom est ajouté dans la zone d'index
- Lorsque l'on valide un fichier, il est ajouté au répertoire Git.

# Introduction

---

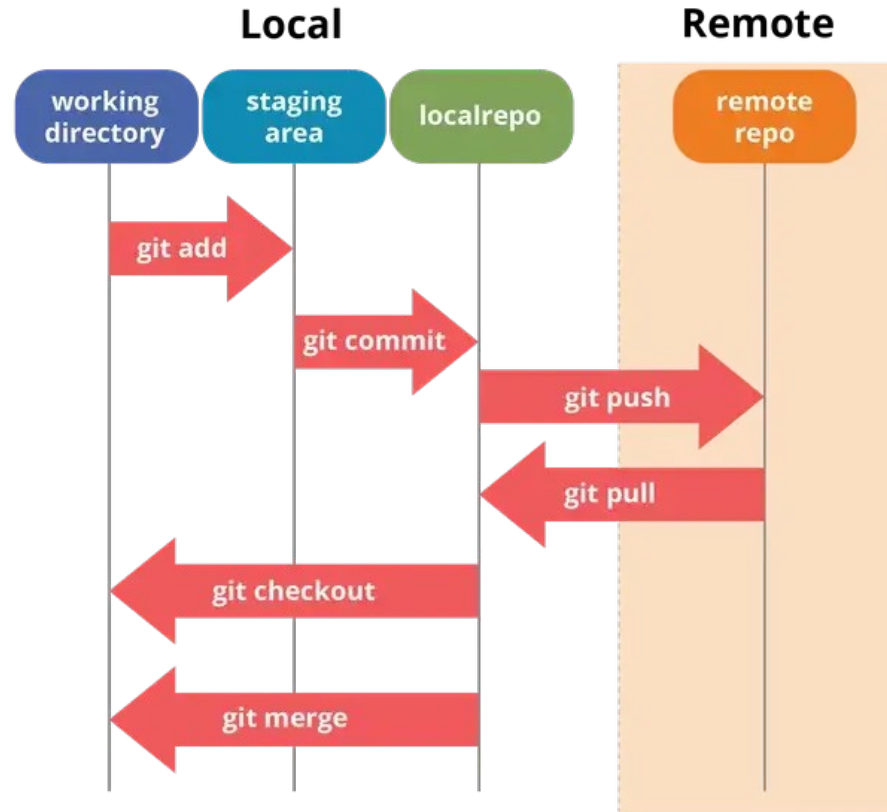
## ▣ GitHub

- Service en ligne qui permet d'héberger des dépôts ou repo Git.
- Une grande partie des dépôts hébergés sur GitHub sont publics, ce qui signifie que n'importe qui peut télécharger le code de ces dépôts et contribuer à leur développement en proposant de nouvelles fonctionnalités.

## ▣ Pour récapituler :

- Git est un logiciel de gestion de version
- GitHub est un service en ligne d'hébergement de dépôts Git qui fait office de serveur central pour ces dépôts.

# Mémo-technique

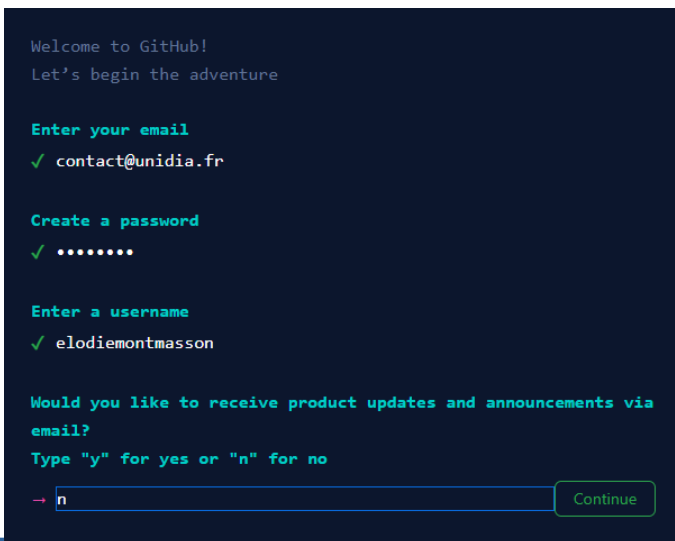


# Installation et configuration

# Installation et configuration

## ❑ Inscription à GitHub

- Ouvrir votre navigateur et allez sur la page :  
<https://github.com/signup?source=login>
- Remplissez le formulaire :



```

Welcome to GitHub!
Let's begin the adventure

Enter your email
✓ contact@unidia.fr

Create a password
✓ .....

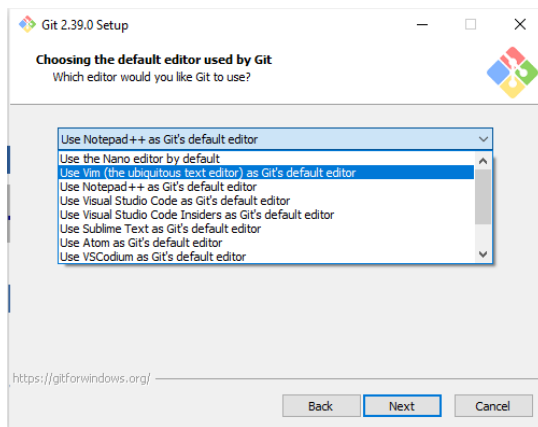
Enter a username
✓ elodiemontmasson

Would you like to receive product updates and announcements via
email?
Type "y" for yes or "n" for no
→ n [Continue]
```

# Installation et configuration

## ❑ Installation

- Allez sur le site officiel : <http://git-scm.com/downloads>
- Téléchargez le fichier, exécutez-le.
- Cliquez sur suivant jusqu'à arriver à cette fenêtre et sélectionnez votre éditeur de texte préféré :



- Puis laissez les options suivantes par défaut

# Installation et configuration

## ❑ Configuration

- Nous allons faire en sorte de lier votre compte Github à votre Git local.
- Lancez le programme Git Bash
- Indiquez votre nom :

```
git config --global user.name "Elodie Montmasson"
```

- Indiquez l'adresse mail de votre inscription :

```
git config --global user.email contact@unidia.fr
```

# Installation et configuration

## ❑ Configuration

### ○ Suivez les instructions :

```
! First copy your one-time code: 83FA-80A2
Open this URL to continue in your web browser: https://github.com/login/device
```

- Rendez-vous sur la page : <https://github.com/login/device>
- Tapez le code indiqué :

Device Activation

Enter the code displayed on your device

8 3 F A - 8 0 A 2

Continue

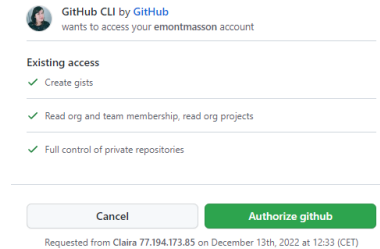
GitHub staff will never ask you to enter your code on this page.



# Installation et configuration

## ■ Configuration

- Cliquez sur « Authorize github »



Vous avez normalement le message suivant :



**Congratulations, you're all set!**

Your device is now connected.

# Installation et configuration

- En retournant sur l'invite de commande, vous devriez retrouver les informations suivantes :

```
! First copy your one-time code: 83FA-80A2
Open this URL to continue in your web browser: https://github.com/login/device
✓ Authentication complete.
✓ Logged in as emontmasson

A new release of gh is available: 2.12.1 → 2.20.2
https://github.com/cli/cli/releases/tag/v2.20.2
```

# Installation et configuration

---

## ❑ Exercice

Installez et configurez Git.

Inscrivez-vous sur Github

# Création et initialisation d'un dépôt local

# Création et initialisation d'un dépôt local

## ❑ Création du dépôt

- Allez dans un répertoire que vous souhaitez versionner
- Cliquez-droit et cliquez sur « Git Bash Here »
- Tapez la commande : `git init` #initialise un dépôt git

Cela crée un sous répertoire `.git` qui contient un ensemble de fichiers qui vont permettre à un dépôt Git de fonctionner.

Git indique que le projet a bien été initialisé et qu'il est vide. C'est tout à fait normal car nous n'avons pas encore versionner les fichiers.

# Création et initialisation d'un dépôt local

---

## ❑ Exercice

Créez un répertoire `git_formation` et initialiser le projet.

# Création et initialisation d'un dépôt local

## ❑ Statuts des fichiers

Après avoir initialisé un projet, la commande affiche le résultat suivant :

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
  full-card-click-main/

nothing added to commit but untracked files present (use "git add" to track)
```

Il indique d'utiliser la commande « git add » pour suivre des fichiers.

# Création et initialisation d'un dépôt local

## □ Indexer des fichiers

- Pour ajouter des fichiers au dépôt Git et constituer ainsi leur historique de modification, il faut d'abord les indexer.

Si vous avez modifié des fichiers qui étaient déjà indexés, vous devez aussi utiliser cette commande pour les préparer au versionning.

Il seront mis de côté dans la partie « stage ».

Pour ce faire, on utilise la commande `git add`.

Options :

- Nom d'un fichier à suivre

```
git add note.txt
```

- Nom d'un répertoire pour suivre tous ses fichiers

```
git add .
```

- Un "fileglob" pour ajouter tous les fichiers correspondant au schéma fourni.

```
git add *.php
```



# Création et initialisation d'un dépôt local

## □ Indexer des fichiers

Après avoir exécuté la commande pour indexer un répertoire, on peut utiliser la commande `git status` pour vérifier l'état des fichiers :

```
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   .DS_Store
    new file:   404.html
    new file:   LICENSE.md
    new file:   README.md
    new file:   apple-touch-icon-114x114-precomposed.png
```

Ils sont à présent considérés comme des nouveaux fichiers et prêts à être enregistrés au dépôt.

Cette commande est utilisée à chaque fois qu'il y a une sauvegarde de travail à faire.

# Création et initialisation d'un dépôt local

---

## ❑ Exercice

Créer un fichier README.md

Indexer le fichier

# Création et initialisation d'un dépôt local

## □ Enregistrer le travail dans le dépôt local (commit)

Il est possible à présent d'enregistrer les fichiers présents dans le stage, avec un auteur, une date, une heure et un message d'enregistrement.

On parle alors pour ce faire de la commande git commit

## □ Options

- Sans option : l'éditeur de texte que vous avez sélectionné pendant l'installation apparaît. Il faut spécifier un message.
- -m : permet d'indiquer directement le message du commit.

**Attention !** Il est important d'écrire un message clair pour indiquer le travail effectué.

Cette commande est utilisée à chaque fois qu'il y a une sauvegarde de travail à faire.

```
git commit -m "initialisation du projet"
```

En tapant la commande git status après git commit, nous pouvons constater qu'il n'y a plus de fichiers à « commit »

```
$ git status
On branch master
nothing to commit, working tree clean
```

# Création et initialisation d'un dépôt local

---

## ❑ Exercice

Enregistrer le travail en précisant le message suivant : « Initialisation du projet avec explications détaillées dans le fichier README.md »

# Création et initialisation d'un dépôt local

---



Le projet a été initialisé localement.

Il faut donc maintenant l'enregistrer sur un serveur.

Cela permettra de le récupérer depuis un autre poste mais aussi de le partager avec d'autres développeurs.

# Création d'un dépôt Github et clonage

# Création d'un dépôt github et clonage

## □ Création d'un dépôt Github

Tout d'abord, il faut créer et configurer le dépôt sur le site de Github.

Aller sur l'onglet « Repositories » et cliquez sur « new ».

Il faut renseigner les champs suivants :

- Le nom du dépôt : on prend généralement le nom du dossier de notre projet
- La description
- La visibilité :
  - Si vous souhaitez développer un projet ouvert à tous, cliquez sur public. N'importe qui pourra récupérer votre projet.
  - Si vous souhaitez développer un projet ouvert à une équipe, par exemple, cliquez sur privé.
- Vous pouvez initialiser le projet avec différents fichiers :
  - Un fichier de documentation readme que nous avons déjà créé.
  - Un fichier .gitignore qui permet d'indiquer des fichiers à ignorer donc à ne pas versionner dans le projet. Il s'agit souvent de bibliothèques. En effet, elles sont souvent définies dans un fichier de configuration et installées par la suite. Le fait de ne pas les versionner permet d'alléger considérablement le projet.  
Il peut être généré selon un type de projet précis comme symfony.
- Vous pouvez aussi choisir une licence pour le développement

# Création d'un dépôt github et clonage

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner \*

emontmasson ▾

Repository name \*

example\_card.git ✓

Great repository names are short and descriptive. Your new repository will be created as example\_card. t shiny-telegram?

Description (optional)

Exemple de cartes pour un site en HTML et CSS

☐ Public

Anyone on the internet can see this repository. You choose who can commit.

☒ Private

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☐ Add a README file

This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore

Choose which files not to track from a list of templates. [Learn more.](#)

.gitignore template: None ▾

Choose a license

A license tells others what they can and can't do with your code. [Learn more.](#)

License: None ▾

 You are creating a private repository in your personal account.

Create repository



# Création d'un dépôt github et clonage

- ❑ Liaison entre le dépôt local et distant

Il faut à présent lier votre dépôt local à Github.

Il faut exécuter la commande suivante en remplaçant « login » et « name\_repo.git » :

```
$ git remote add origin https://github.com/login/name_repo.git
```

# Création d'un dépôt github et clonage

---

## ❑ Exercice

Liez votre projet à Github

# Création d'un dépôt github et clonage

- ❑ Enregistrer votre travail sur Github

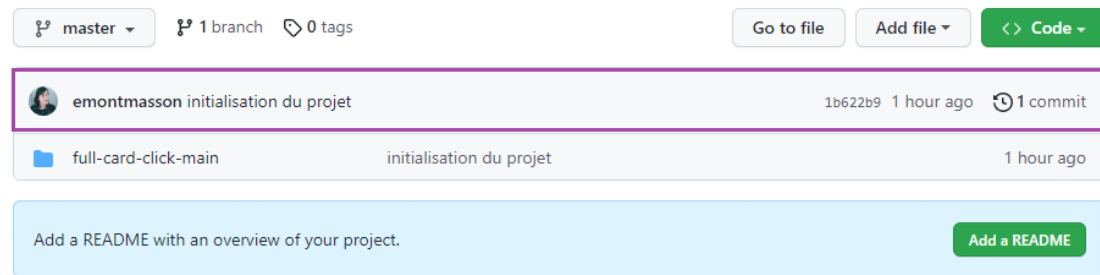
Il faut à présent mettre les modifications apportées en local sur le serveur :

```
git push origin master
```

# Création d'un dépôt github et clonage



Allez sur votre navigateur et ouvrez votre projet sur Github.



Vous pouvez constater que les sources de votre projet sont bien présentes avec les informations du premier commit.

# Création d'un dépôt github et clonage

---

## ❑ Exercice

Enregistrez votre fichier sur GitHub.

# Création d'un dépôt github et clonage

---

## ❑ Partager son projet

Pour permettre à vos collaborateurs d'accéder à un projet, il faut les inviter.

Pour ce faire, allez sur le projet sur Github et cliquez sur

1. « Settings »
2. « Collaborators »
3. « Add people ».

# Création d'un dépôt github et clonage

The screenshot shows the GitHub repository settings page for 'emontmasson / example\_card'. The repository is marked as 'Private'. The navigation bar at the top includes links for Code, Issues, Pull requests, Actions, Projects, Security, Insights, and Settings (highlighted with a purple box and labeled '1.').

On the left sidebar, the 'Access' section is expanded, and 'Collaborators' is selected (highlighted with a purple box and labeled '2.').

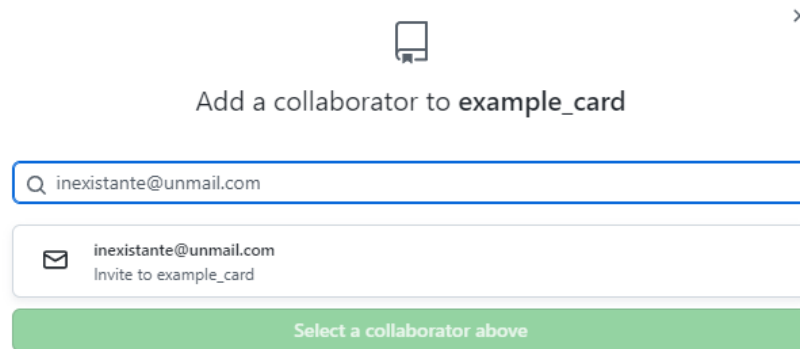
The main content area is titled 'Who has access'. It shows two access methods: 'PRIVATE REPOSITORY' (which is currently selected) and 'DIRECT ACCESS'. The 'PRIVATE REPOSITORY' section states: 'Only those with access to this repository can view it.' and includes a 'Manage' link. The 'DIRECT ACCESS' section states: '0 collaborators have access to this repository. Only you can contribute to this repository.'

Below this, the 'Manage access' section is visible, showing a message: 'You haven't invited any collaborators yet' with a lock icon. A green 'Add people' button is highlighted with a purple box and labeled '3.'.

# Création d'un dépôt github et clonage

Vous pouvez saisir un utilisateur en le recherchant avec son compte github, son nom ou une adresse mail.

Si la personne n'a pas de compte Github, il est tout à fait possible de l'ajouter tout de même.



×

Add a collaborator to `example_card`

Q inexistante@unmail.com

✉ inexistante@unmail.com  
Invite to `example_card`

Select a collaborator above



# Création d'un dépôt github et clonage

---

## ❑ Exercice

Invitez une personne à votre projet

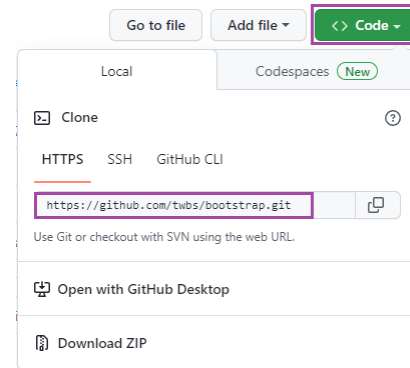
# Création d'un dépôt github et clonage

## ❑ Cloner un projet

Pour récupérer un projet déjà existant, vous devez connaître l'url du projet.

Pour cela, allez sur Github et retrouvez le projet.

Cliquez sur « Code » et copier l'url :



Ensuite, allez sur votre répertoire de travail et taper la commande suivante :

```
git clone https://github.com/twbs/bootstrap.git
```

# Création d'un dépôt github et clonage

---

## ❑ Exercice

Clonez le projet suivant dans votre répertoire de travail : <https://github.com/airbnb/css.git>

# Modification d'un dépôt

# Modification d'un dépôt

- ❑ Ajout ou modification de fichiers
  - Commandes git à utiliser dans l'ordre :
    - git add (diapo 25)
    - git commit (diapo 26)
    - git push (diapo 34)
- ❑ Suppression de fichiers
  - Suppression physique du fichier et dans le dépôt
    - rm nom-du-fichier
    - Git rm nom-du-fichier
  - Conservation physique du fichier mais suppression de son suivi
    - Git rm --cached nom-du-fichier

# Modification d'un dépôt

## ❑ Retrait d'un fichier de la zone d'index

Cas d'un ajout d'un fichier par erreur dans la zone d'index (commande git add).

- Git reset HEAD nom-du-fichier

Le fichier continue d'être suivi mais la version actuelle ne fera pas partie du prochain instantané.

## ❑ Empêcher l'indexation de certains fichiers dans Git

- Fichier .gitignore à créer à la racine du projet
- Concerne les fichiers générés automatiquement, bibliothèques ou fichiers sensibles.
- A chaque ajout d'un fichier répertorié dans le .gitignore, il ne sera pas affiché comme un fichier non indexé suite à l'exécution de la commande git status

## ❑ Renommer un fichier dans Git

- Commande : git mv ancien-nom-fichier nouveau-nom-fichier.

# Modification d'un dépôt

## ❑ Exercice

- Retournez sur votre projet git\_formation.  
Ajoutez des lignes dans le fichier README.md.  
Créez un fichier index.php.  
Créez un fichier .htaccess.  
Sauvegarder votre travail sur le serveur
- Supprimez le fichier .htaccess  
Sauvegardez votre travail sur le serveur
- Renommez le fichier README.md en LISEZMOI.md  
Sauvegardez votre travail sur le serveur

# Historique d'un dépôt



# Historique d'un dépôt

- Consulter la liste des commits

Il est possible de consulter la liste des commits effectués sur un projet avec la commande git log :

```
$ git log
commit 1b622b9dee2c8a6604945bc583edc999ba5c4b05 (HEAD -> master)
Author: Elodie Montmasson <contact@unidia.fr>
Date: Tue Dec 13 17:19:00 2022 +0100

    initialisation du projet
```

Le commit a été enregistré. On voit différentes informations apparaître :

- Le numéro de commit : 1b622b9dee2c8a6604945bc583edc999ba5c4b05
- La branche concernée : master
- L'auteur : Elodie Montmasson
- La date et l'heure du commit : mardi 13 décembre à 17h19
- Le message du commit : Initialisation du projet

# Historique d'un dépôt

- ❑ Écraser et remplacer un commit
  - Cas d'annulation d'une validation (un commit) :
    - Omission de fichiers
    - Mauvaises versions
    - Mauvais message de commit
  - Solution la plus simple :
    - `git commit --amend -m message` : pousse un nouveau commit qui va remplacer le précédent en l'écrasant
  - Exemples :
    - Erreur sur le message de commit :
      - `Git commit --amend -m « Développement version mobile »`

# Historique d'un dépôt

- ❑ Annuler des modifications apportées à un fichier
  - But : Revenir à un état antérieur enregistré d'un projet c'est à dire au dernier commit.
  - Cause : erreurs générés suite à la modification du projet.
  - Commandes :
    - `git checkout --nom-du-fichier`
    - `git restore --nom-du-fichier`

# Historique d'un dépôt

- ❑ Annuler des modifications apportées à un fichier
  - Commande :
    - `git reset --hard HEAD` (cela permet d'aller au dernier commit)
    - `git reset --hard HEAD^2` (cela permet d'aller à l'avant dernier commit)
    - `git reset --hard numero-de-commit` (cela permet d'aller au commit que l'on a précisé en paramètre)

# Historique d'un dépôt

---

## ❏ Exercice

- Récupérez le fichier .htaccess
- Refaire le dernier message de commit en indiquant « ajout des redirections »

# Gestion des branches

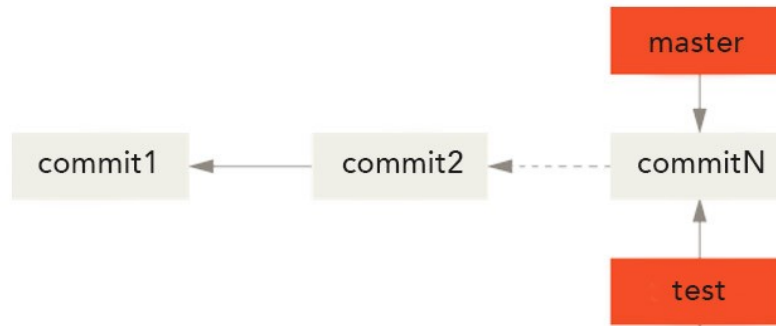
# Gestion des branches

## □ Présentation

- Définition : Copie du projet. Il s'agit d'un pointeur vers un commit.
- But : Permet de tester de nouvelles fonctionnalités sans impacter le projet de base.
- Branche par défaut : master.

## □ Création d'une nouvelle branche

- Commande : `git branch nom-de-la-branche`.
- Création d'un nouveau pointeur vers le dernier commit effectué (le commit courant).
- A ce stade, branches créées :
  - Master
  - Celle qui vient d'être tout juste créé.



# Gestion des branches

## ❑ Création d'une nouvelle branche

- Pour déterminer quel pointeur vous utilisez, c'est-à-dire sur quelle branche vous vous trouvez, Git utilise un autre pointeur spécial appelé HEAD.

HEAD pointe sur la branche master par défaut.

Notez que la commande `git branch` permet de créer une nouvelle branche mais ne déplace pas HEAD.

Nous allons donc devoir déplacer explicitement HEAD pour indiquer à Git qu'on souhaite basculer sur une autre branche.



# Gestion des branches

- ❑ Basculer entre les branches
  - Commande : `git checkout nom-de-la-branche`
- ❑ Créer et basculer sur la nouvelle branche
  - Commande : `git checkout -b nom-de-la-branche`
- ❑ Récupérer toutes les branches du dépôt
  - Commande : `git fetch --all`
- ❑ Récupérer une branche du dépôt non présente sur l'espace de travail
  - Commande : `git fetch origin nom-de-la-branche`

# Gestion des branches

---

## ❏ Exercice

Créez et basculez sur une branche nommée « dev001 ».

# Gestion des branches

---

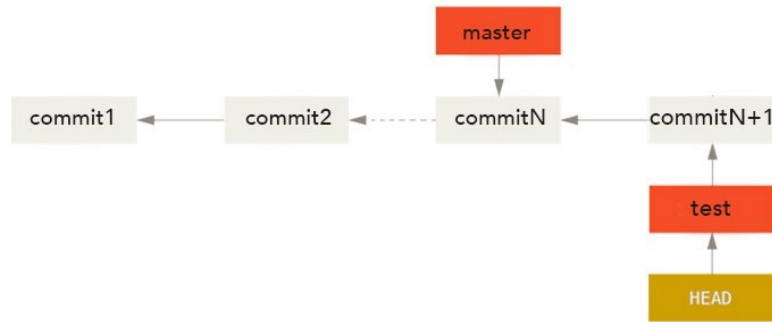
## ❑ Fusion et rebase

- Permet d'intégrer de nouvelles fonctionnalités au sein du logiciel
- Fusionner des branches
  - On parle de divergence de branche quand deux branches possèdent :
    - un ancêtre commit en commun
    - pointent chacune vers de nouveaux commits qui peuvent correspondre à des modifications différentes d'un même fichier du projet.

# Gestion des branches

## □ Fusion et rebase

### ○ Fusionner des branches : exemple



### ○ Explication

- La branche test pointe sur un commit commitN+1
- une branche master qui pointe sur un commit commitN
- commitN est l'ancêtre direct de commitN+1 et il n'y a donc pas de problème de divergence.

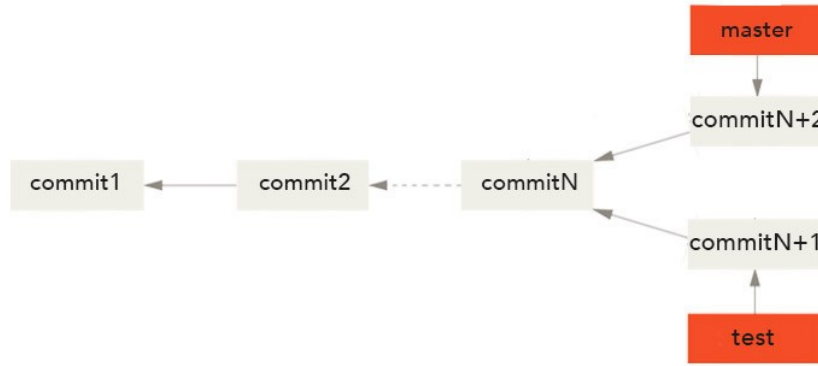
### ○ Commandes :

- Git checkout master (pour se placer sur la branche master)
- Git pull origin master (pour récupérer les dernières mises à jour effectuées sur master)
- Git merge test (pour fusionner la branche test à master)
- Git push origin master (pour valider la fusion)
- Git branch -d test (pour supprimer la branche test)

# Gestion des branches

## ▣ Fusion et rebase

### ○ Fusionner des branches : autre exemple



### ○ Explication

- Pour fusionner deux branches ici on va à nouveau se placer dans la branche dans laquelle on souhaite fusionner puis effectuer un git merge.
- Git réalise une fusion en utilisant 3 sources :
  - le dernier commit commun aux deux branches
  - le dernier commit de chaque branche.
- Création d'un nouvel instantané dont le contenu est le résultat de la fusion des commits de chaque branche.
- Notez que dans le cas d'une fusion à trois sources, il se peut qu'il y ait des conflits.
- Cela va être notamment le cas si une même partie d'un fichier a été modifiée de différentes manières dans les différentes branches. Dans ce cas, lors de la fusion, Git nous alertera du conflit et nous demandera de le résoudre avant de terminer la fusion des branches.

# Gestion des branches

## ❑ Fusion et rebase

### ○ Fusionner des branches : autre exemple

#### ○ Commandes

- Git status (pour voir précisément quels fichiers sont à l'origine du conflit).

```
$ git status
On branch dev001
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:   css/styleMobile.css

no changes added to commit (use "git add" and/or "git commit -a")
```

Imaginons par exemple que nos deux branches possèdent un fichier styleMobile.css et que les deux fichiers possèdent des textes différents.

# Gestion des branches

## ▣ Fusion et rebase

### ○ Fusionner des branches : autre exemple

#### ○ Commandes

Git va automatiquement “fusionner” les contenus des deux fichiers en un seul qui va en fait contenir les textes des deux fichiers de base à la suite l'un de l'autre avec des indicateurs de séparation.

```
a.menu {
<<<<<< HEAD
  color:red;
=====
  color:green;
>>>>>> master
  text-align: center;
  text-decoration: underline;
}

a {
  color:green;
}
```

- On peut alors ouvrir le fichier à la main et choisir ce qu'on conserve (en supprimant les parties qui ne nous intéressent pas par exemple).

```
]a.menu {
  color:red;
  text-align: center;
  text-decoration: underline;
;}

[a {
  color:green;
}
```

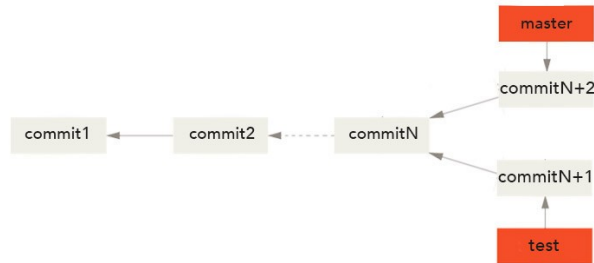
- git add pour marquer le conflit comme résolu. On n'aura alors plus qu'à effectuer un git commit pour terminer le commit de fusion.

# Gestion des branches

## ▣ Fusion et rebase

### ○ rebase

#### ○ Exemple de branche qui divergent



### ○ Explication

Plutôt que d'effectuer une fusion à trois sources, on va pouvoir rebaser les modifications validées dans commitN+1 dans notre branche master.

### ○ Commande

- Git checkout master
- Git pull origin master
- Git rebase test
- Git push origin master
- Git branch -d test



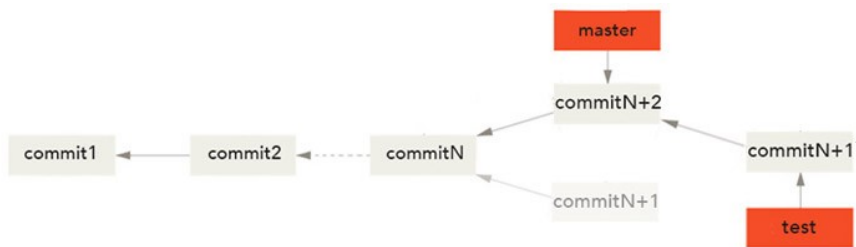
# Gestion des branches

## ❑ Fusion et rebase

### ○ rebase

#### ○ Explication

Dans ce cas, Git part à nouveau du dernier commit commun aux deux branches (l'ancêtre commun le plus récent) puis récupère les modifications effectuées sur la branche qu'on souhaite rapatrier et les applique sur la branche vers laquelle on souhaite rebaser notre travail dans l'ordre dans lequel elles ont été introduites.



- Le résultat final est le même qu'avec une fusion mais l'historique est plus clair puisque toutes les modifications apparaissent en série même si elles ont eu lieu en parallèle. Rebaser rejoue les modifications d'une ligne de commits sur une autre dans l'ordre d'apparition, alors que la fusion joint et fusionne les deux têtes.

# Gestion des branches

## ▣ Exercice

Allez sur la branche master.

Editez le fichier index.php en ajoutant : 

```
<?php  
echo "hello word";  
?>
```

Validez votre travail.

Allez sur la branche dev001.

Editez le fichier index.php en mettant : 

```
<?php  
echo "hello la compagnie";  
?>
```

Validez votre travail

- Fusionnez la branche master sur dev001.

# GitFlow

# GitFlow

## ❑ Présentation

- Un modèle de branche
- Un plugin git pour automatiser le travail

## ❑ Fonctionnement

- Projet basé sur deux branches :
  - Master : miroir de notre production.
  - Develop : centralise toutes les nouvelles fonctionnalités qui seront livrées dans la prochaine version

Ces deux branches sont strictement interdites en écriture aux développeurs.

- Branches de travail :
  - Feature : pour développer une fonctionnalité
  - Release : pour réunir un ensemble de fonctionnalités afin de préparer une nouvelle version du logiciel. Elle dédiée aux corrections de bugs, à la génération de documentation et à d'autres tâches axées sur la livraison.  
Une fois qu'elle est prête, la branche release est mergée dans la branche main, et Git lui attribue un numéro de version.  
Il convient en outre de faire à nouveau un merge de la livraison dans la branche develop, qui a certainement progressé depuis le début de la livraison.
  - Hotfix : pour réparer des fonctionnalités

# GitFlow

## ❑ Initialisation

Allez sur votre projet et allez sur la branche master et lancez :

```
$ git flow init
```

Vous pouvez considérer que les réponses par défaut sont toutes correctes. Si la branche develop existe déjà, elle sera utilisée, le processus la créera sinon.

# GitFlow

## ❑ Création d'une branche de fonctionnalité

- Sans les extensions git-flow :

```
git checkout develop  
git checkout -b feature_branch
```

- Avec les extensions git-flow :

```
git flow feature start feature_branch
```

Continuez votre travail et utilisez Git comme vous le feriez normalement.

# GitFlow

- ❑ Terminer une branche de fonctionnalité
  - Lorsque vous avez terminé le travail de développement sur la fonctionnalité, l'étape suivante consiste à merger la branche feature dans la branche develop.
  - Sans les extensions git-flow :
- Avec les extensions git-flow :

```
git checkout develop  
git merge feature_branch
```

```
git flow feature finish feature_branch
```

# GitFlow

- ❑ Préparer une nouvelle version
  - Sans les extensions git-flow :

```
git checkout develop  
git checkout -b release/0.1.0
```

- Avec les extensions git-flow :

```
$ git flow release start 0.1.0  
Switched to a new branch 'release/0.1.0'
```



# GitFlow

## ❑ Préparer la mise en production

- Sans les extensions git-flow :

```
git checkout main  
git merge release/0.1.0
```

- Avec les extensions git-flow :

```
git flow release finish '0.1.0'
```

# GitFlow

## ❑ Création d'une hotfix

- Sans les extensions git-flow :

```
git checkout main  
git checkout -b hotfix_branch
```

- Avec les extensions git-flow :

```
$ git flow hotfix start hotfix_branch
```

# GitFlow

## ❑ Préparer la mise en production de la hotfix

- Sans les extensions git-flow :

```
git checkout main
git merge hotfix_branch
git checkout develop
git merge hotfix_branch
git branch -D hotfix_branch
```

- Avec les extensions git-flow :

```
$ git flow hotfix finish hotfix_branch
```