

Git - Gérer le versioning

GIT-VER

m2iinformation.fr



Déroulement de la formation

- Jour 1
 - Présentation de Git
 - Comprendre les principes de Git
 - Prise en main
 - Travailler en équipe
 - Gestion des branches
- Jour 2
 - Compléments

PRÉSENTATION DE GIT



Présentation de Git (1/3) - Présentation et utilité

- Logiciel de gestion de versions
 - Permet de gérer l'évolution du contenu d'une arborescence via une architecture client/serveur
 - Sous licence GNU (libre et open-source)
- Pourquoi l'utiliser ?
 - Suivre les changements d'un projet
 - Gérer les conflits d'édition
 - Réaliser des sauvegardes régulières

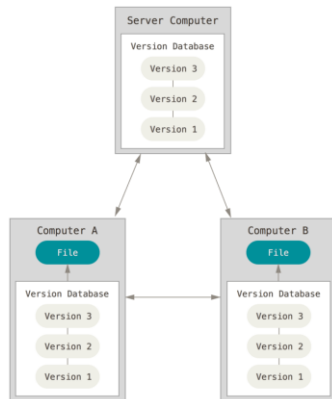
Présentation de Git (2/3) - Comparaison avec subversion (SVN)

GIT

Logiciel de gestion de versions décentralisé

« copie locale »
dépôt à part entière

Permet à ce titre de faire des
« commits » locaux

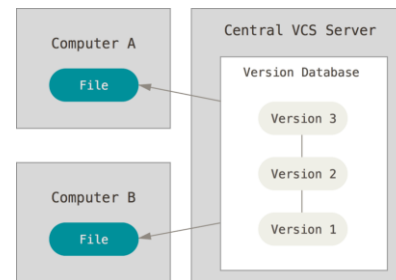


SVN

Logiciel de gestion de versions centralisé

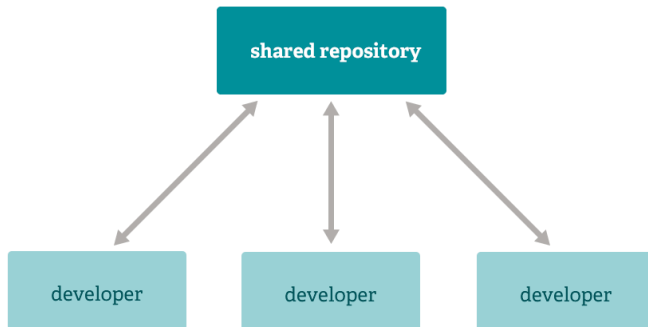
« copie locale »
copie en lecture du dépôt

Les commits sont envoyés
directement au serveur

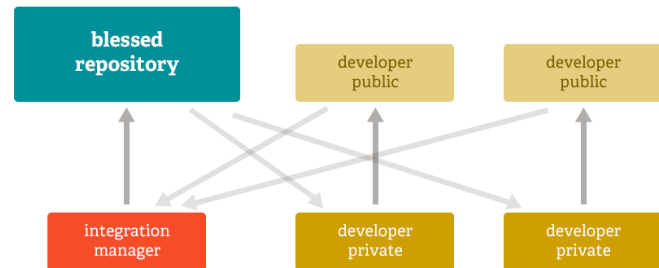


Présentation de Git (3/3) - Aperçu des flux de travaux possibles

« Centralisé »



« Responsable »
ayant autorité



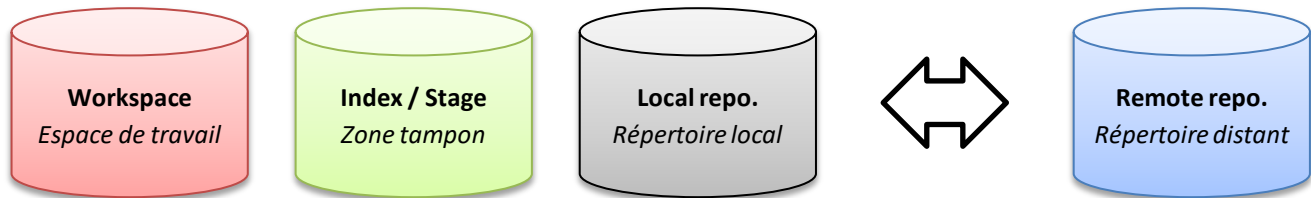
COMPRENDRE LES PRINCIPES DE GIT

The background is a solid red color. In the lower half, there are stylized silhouettes of three people. The person in the middle has a large number '2' on their chest. The silhouettes are a darker shade of red than the background.

2

Présentation des concepts de Git - Les différents flux et la décentralisation

- **Remote repository** Historique du projet sur le serveur (*dossier .git*)
- **Local repository** Historique du projet sur le client (*dossier .git*)
- **Index / Stage** Liste des fichiers indexés
- **Workspace** Liste des fichiers modifiés et/ou supprimés



Présentation des concepts de Git - Aperçu des flux de travaux possibles

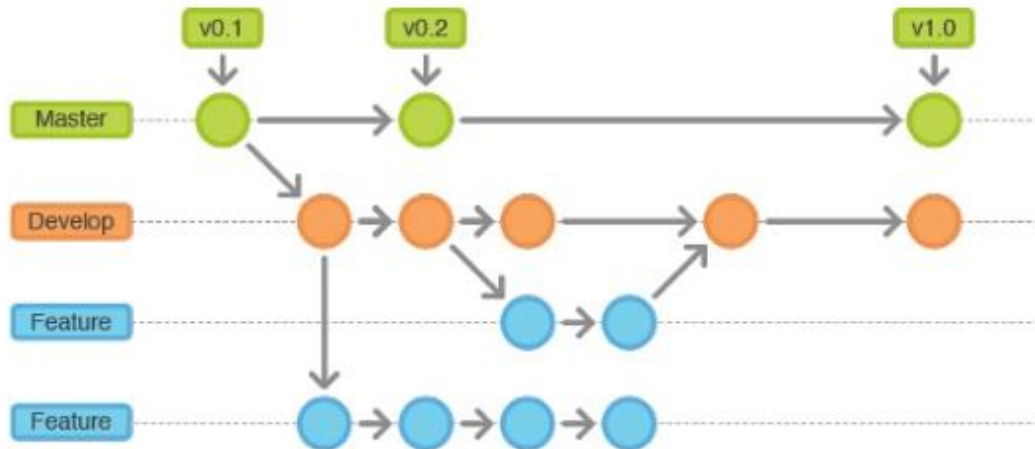
- **untracked** Nouveaux fichiers (*non connus de Git*)
- **modified** Fichiers connus de Git et modifiés (*dans le workspace*)
- **staged** Fichiers connus de Git, modifiés et indexés
- **stable** Fichiers connus de Git et non modifiés
- **deleted** Fichiers connus de Git et supprimés (*dans le workspace*)

HEAD : pointeur sur la référence de la branche actuelle, qui est à son tour un pointeur sur le dernier *commit* réalisé sur cette branche



Présentation des concepts de Git - Les branches

- Permet de créer des sous-espaces de travail
 - Chaque branche peut évoluer de manière séparée
 - On peut basculer d'une branche à l'autre « à tout moment »

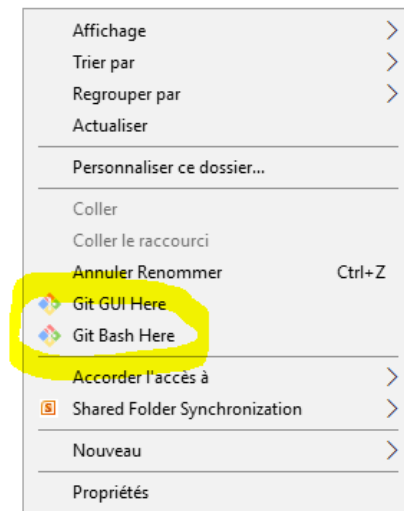


PRISE EN MAIN



Prise en main (1/2) - Installation et configuration

- Installation sous Windows <https://git-scm.com>
 - Git Bash *émulateur de console Unix*
 - Git GUI *interface graphique*
 - Intégration automatique dans Windows
- Configuration
 - **git config** --global user.name "votre_pseudo"
 - **git config** --global user.email moi@email.com
 - **git config** --list



Prise en main (1/2) - Nouveautés

- (v2.25) Nouvelles commandes : **git restore / git switch**
 - Ecraser des modifications
 - **git checkout file** **git restore file**
 - Changer de branche
 - **git checkout branch** **git switch branch**
 - **git checkout -b branch** **git switch -c branch**
- Revenir en arrière dans l'historique
 - **git checkout ab12c3**

Prise en main (2/2) - Commandes principales

- Commandes pour démarrer
 - **git init** *Création du dépôt dans le répertoire courant*
 - **git init mon-projet** *Création du dépôt dans le répertoire mon-depot*
 - **git clone https://...** *Clonage d'un dépôt depuis un serveur distant*
 - **git status** *Affiche l'état du dépôt*

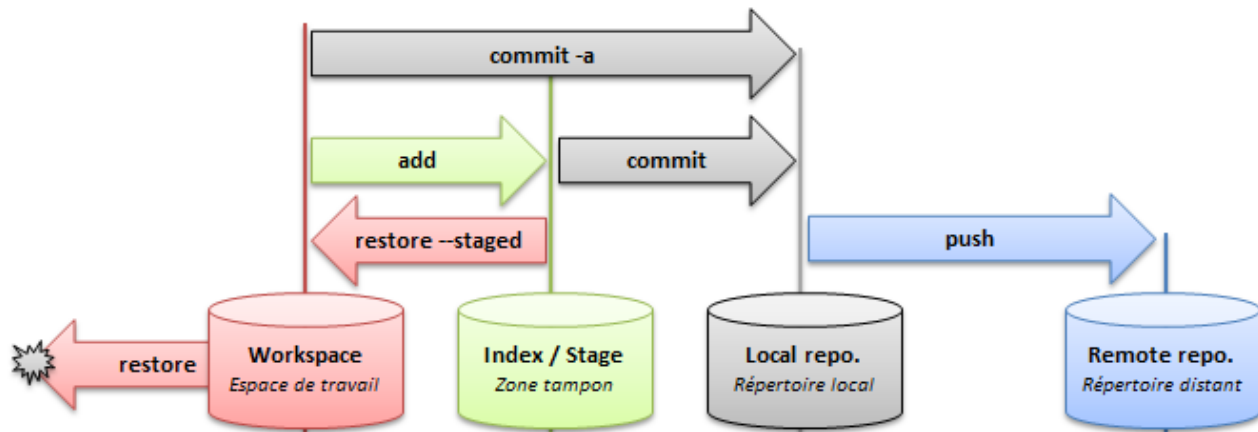
```
F2000@F2000-PC MINGW64 /d/www/formation (master)
$ git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
```

Prise en main (2/2) - Commandes principales

- Commandes pour envoyer des modifications
 - **git add** *Ajoute des fichiers dans l'index local*
 - **git restore --staged** *Retire des fichiers de l'index local*
 - **git rm [--cached]** *Supprime des fichiers de l'index local*
- **git commit** *Compacte l'index local au sein d'un « commit »*
- **git push** *Envoie les « commits » locaux sur le serveur*



Créer un premier dépôt en local.

Créer un premier fichier
« hello.txt » contenant
« Hello world ! »

Ajouter et intégrer ce fichier
dans un « commit ».

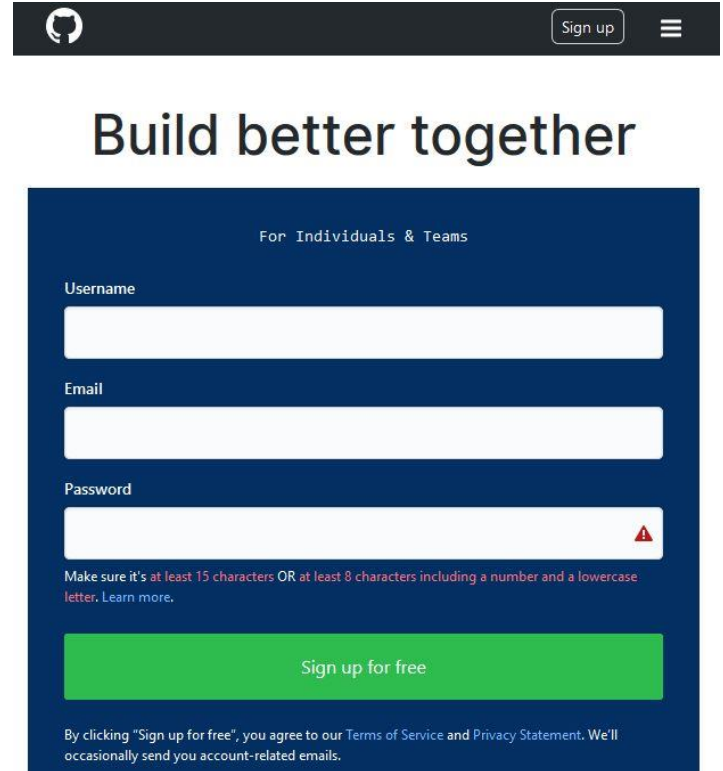
Tester la commande « *git push* »

Service web

Hébergement et de gestion de projets / code source (via Git)

Fonctionnalités annexes

- « Bugtracker »
- « Wiki »



The screenshot shows the GitHub sign-up page. At the top, there's a dark header with the GitHub logo on the left, a 'Sign up' button in the center, and a hamburger menu icon on the right. Below the header, the main heading 'Build better together' is displayed in a large, bold font. Underneath this, it says 'For Individuals & Teams'. The sign-up form consists of three input fields: 'Username', 'Email', and 'Password'. The 'Password' field has a red warning icon on the right. Below the 'Password' field, there's a note: 'Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. Learn more.' At the bottom of the form is a large green button labeled 'Sign up for free'. Below the button, there's a small disclaimer: 'By clicking "Sign up for free", you agree to our Terms of Service and Privacy Statement. We'll occasionally send you account-related emails.'

Exercice

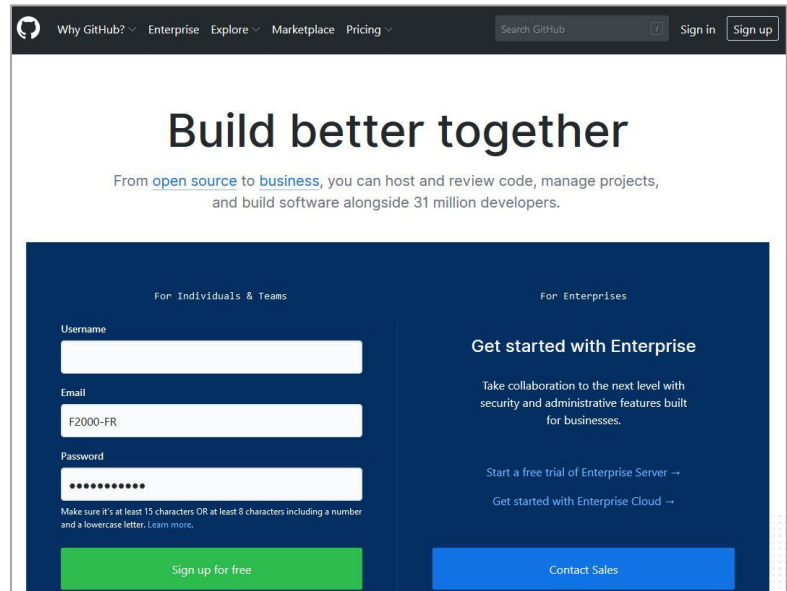
Créer un dépôt distant
grâce à Github.

Envoyer le commit
précédent sur le dépôt
distant.

Astuce :

git remote add ...

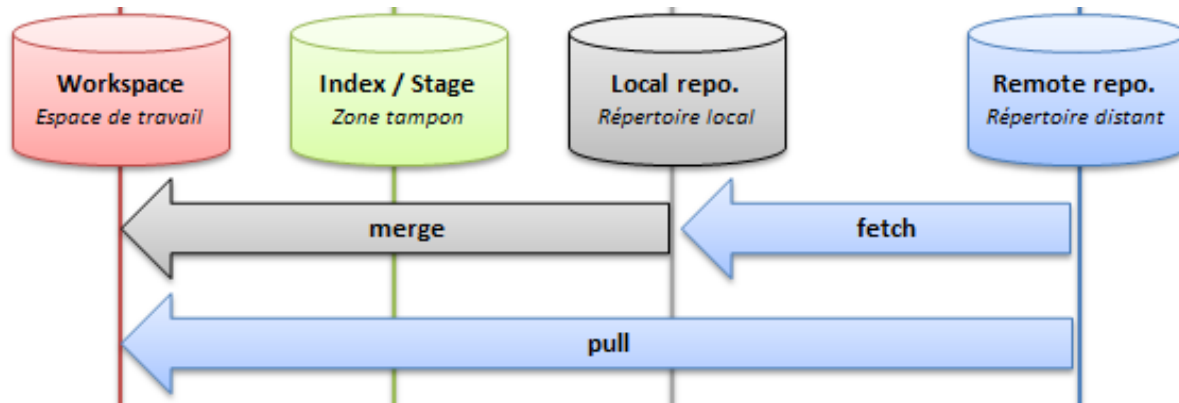
Pour ajouter un dépôt
distant



The screenshot shows the GitHub homepage with a dark header bar containing navigation links: 'Why GitHub?', 'Enterprise', 'Explore', 'Marketplace', and 'Pricing'. A search bar and 'Sign in'/'Sign up' buttons are on the right. The main heading is 'Build better together', followed by a subtext: 'From [open source](#) to [business](#), you can host and review code, manage projects, and build software alongside 31 million developers.' Below this is a sign-up form for 'Individuals & Teams' with fields for 'Username', 'Email' (containing 'F2000-FR'), and 'Password'. A note specifies password requirements: 'Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more.](#)' A green 'Sign up for free' button is at the bottom. To the right, a section for 'Enterprises' titled 'Get started with Enterprise' describes collaboration features and offers links to 'Start a free trial of Enterprise Server' and 'Get started with Enterprise Cloud', with a blue 'Contact Sales' button at the bottom.

Prise en main (2/2) - Commandes principales

- Commandes pour recevoir des modifications
 - **git pull ...** *Récupère des modifications depuis le serveur et les applique sur le répertoire de travail*
 - **git fetch ...** *Récupère des modifications depuis le serveur*
 - **git merge** *Applique les modifications sur le répertoire de travail*



Exercice

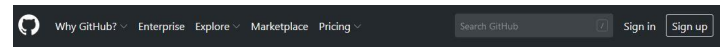
Sur Github, créer un fichier README.md et créer un commit.

Essayer les commandes suivantes :

- **git status**
- **git push**

Puis dans un second temps

- **git fetch**
- **git status**
- **git merge**



Build better together

From [open source](#) to [business](#), you can host and review code, manage projects, and build software alongside 31 million developers.

The image shows the GitHub sign-up page. It is divided into two main sections: 'For Individuals & Teams' and 'For Enterprises'. The 'For Individuals & Teams' section contains a form with fields for 'Username', 'Email' (pre-filled with 'F2000-FR'), and 'Password'. Below the password field is a note: 'Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. Learn more.' At the bottom of this section is a green button labeled 'Sign up for free'. The 'For Enterprises' section is titled 'Get started with Enterprise' and includes the text 'Take collaboration to the next level with security and administrative features built for businesses.' It has two links: 'Start a free trial of Enterprise Server ->' and 'Get started with Enterprise Cloud ->'. At the bottom of this section is a blue button labeled 'Contact Sales'.

TRAVAILLER EN ÉQUIPE



Voir les différences : **git diff**

- Lorsque l'on modifie plusieurs fichiers, il peut être utile de réafficher les modifications effectuées
 - **git diff**
 - affiche les modifications des fichiers modifiés, non indexés
 - **git diff --cached**
 - affiche les modifications des fichiers modifiés et indexés
 - **git diff HEAD**
 - affiche les modifications par rapport au dépôt local
- On peut également afficher des modifications entre des branches ou des commits
 - **git diff *master origin/master***
 - affiche les modifications de la branche locale par rapport à la branche distante (« locale »)
 - **git diff *master develop***
 - affiche les modifications de la branche locale « master » par rapport à la branche locale « develop »
 - **git diff *hash1 hash2***
 - affiche les modifications entre les deux commits indiqués

Voir l'historique des changements : **git log** / **git show**

- Après de nombreux « *commits* », il peut être intéressant d'afficher l'historique des modifications
 - **git log**
 - affiche les différents commits effectués sur le dépôt
 - **git log -p**
 - affiche le détail des différents commits effectués sur le dépôt
 - **git show *hash***
 - Affiche le détail d'un commit spécifique grâce à son « hash »

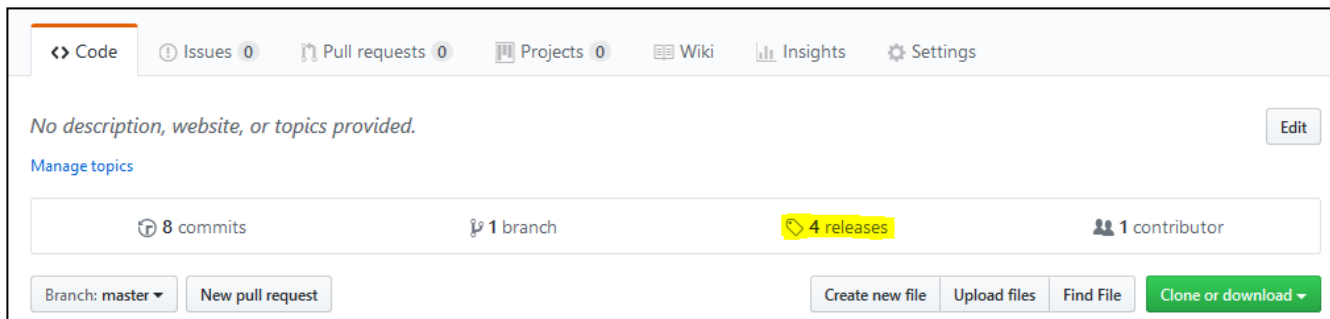
Etiqueter des versions : **git tag**

- De temps en temps, il peut être utile de « taguer » un état du projet (ex: v1, v2, etc.)
 - **git tag**
 - affiche les étiquettes existantes
 - **git tag v1 -m « Version 1 »**
 - crée l'étiquette « v1 », *sur le dernier commit*, avec comme message « Version 1 »
 - **git tag v1 hash -m « Version 1 »**
 - crée l'étiquette « v1 », *sur le commit spécifié*, avec comme message « Version 1 »
 - **git show v1**
 - permet d'afficher le détail de l'étiquette
 - **git push origin v1**
 - permet d'envoyer l'étiquette sur le serveur
 - « *git push origin --tags* » pour envoyer toutes les étiquettes

Exercice

Créer un tag localement et l'envoyer sur le dépôt distant (Github)

Y accéder ensuite sur Github via **Code > releases**



Gestion des conflits

- Survient dès lors qu'un même fichier a été modifié par des « *commits* » différents sur des lignes communes
 - Soit Git pourra corriger les conflits automatiquement
 - Soit Git vous donnera la main pour les corriger

```
F2000@F2000-PC MINGW64 /d/www/formation (master)
$ git merge
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.

F2000@F2000-PC MINGW64 /d/www/formation (master|MERGING)
$ cat README.md
<<<<<<< HEAD
# Projet Git

Ceci est une formation M2i.
=====
# FooBar|

FooBar is a Python library for dealing with word pluralization.
>>>>>>> refs/remotes/origin/master
```

Exercice

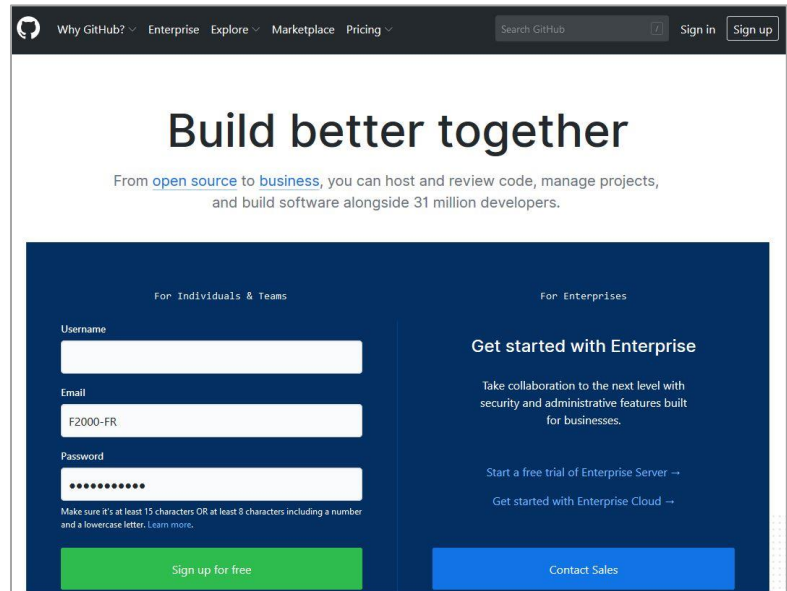
Sur Github, modifier le fichier README.md et créer un commit.

Modifier également le fichier README.md en local et créer un commit.

Effectuer les commandes suivantes :

- **git fetch**
- **git status**
- **git merge**

Corriger le conflit



Annuler des actions (1/2) : sur le dépôt local

- Modifier le dernier commit non propagé
 - La commande « **git commit --amend** » permet de modifier un commit local (sur le « local repository »)
- Annuler le dernier commit non propagé
 - La commande « **git reset HEAD~n** » permet d'annuler N commits locaux et remet les modifications dans le « workspace »
 - L'option « --soft » garde l'indexation précédemment réalisée
 - L'option « --hard » efface définitivement les modifications
- Désindexer un fichier
 - La commande « **git restore --staged** » ou « **git reset HEAD ...** » permet de désindexer les fichiers spécifiés (ou tout l'index courant pour « git reset HEAD »)
- Réinitialiser un fichier modifié
 - La commande « **git restore** » ou « **git checkout** » permet de réinitialiser toutes les modifications locales d'un fichier.

Exercice

- Tester l'amendement de commit
 - Créer un commit C1, puis le modifier en C1'
- Créer 2 nouveaux commits et les annuler
 - Créer C2 et C3, puis revenir à C1'
- Envoyer le résultat (C1') sur le serveur

Astuce : vérifier l'état courant via « *git log* »

Annuler des actions (2/2) : sur le dépôt distant

- Annuler le dernier commit propagé sur le serveur
 - « **git reset --hard HEAD~n** » revient en arrière de N commits
 - « **git push** » refusé par Git si les commits ont été propagés sur le serveur
 - « **git push -f** » permet de pousser « en force » mais est très dangereux à utiliser puisque cela écrase l'historique du serveur
- Bonne méthode : appliquer un commit « inverse »
 - La commande « **git revert hash** » permet d'annuler un commit présent sur le serveur (ou créant son commit inverse). Il faut ensuite propager ce commit sur le serveur.
 - « **git revert HEAD~3..HEAD** » permet d'annuler les trois derniers commits (et va créer 3 commits inverses)

Exercice

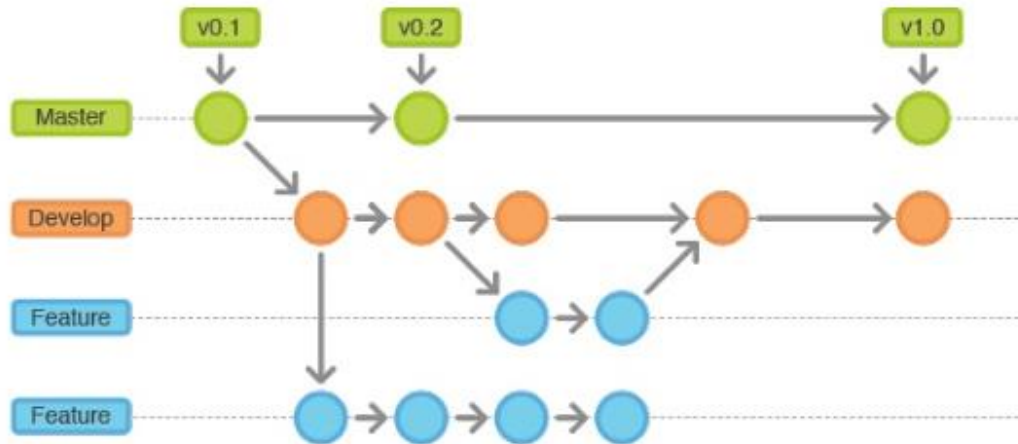
- Annuler le commit C1' précédemment envoyé sur le serveur (méthode 1)
 - **git reset --hard [...]**
- Refaire un commit C1' (et l'envoyer sur le serveur) puis l'annuler
 - **git revert [...]**

GESTION DES BRANCHES



Gestion des branches (1/3) - Présentation et utilité

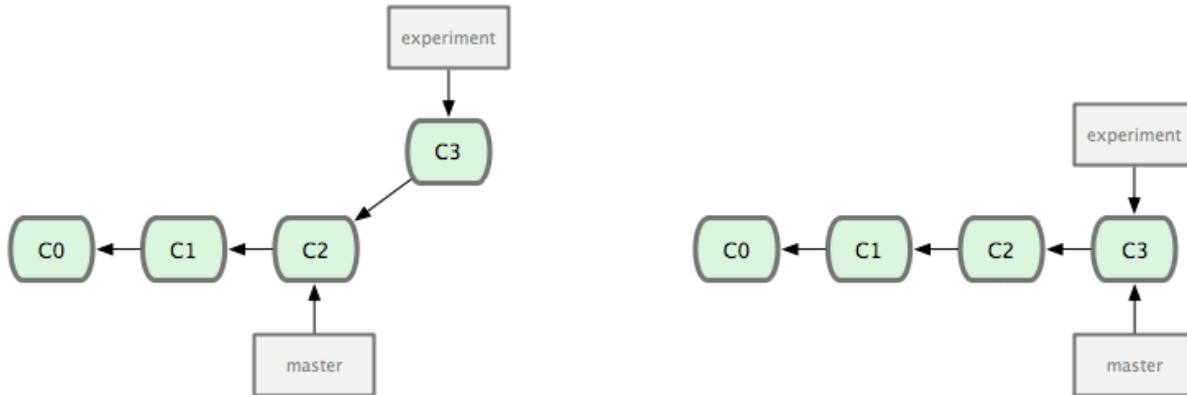
- Permet de créer des sous-espaces de travail
 - Chaque branche peut évoluer de manière séparée
 - On peut basculer d'une branche à l'autre « à tout moment »



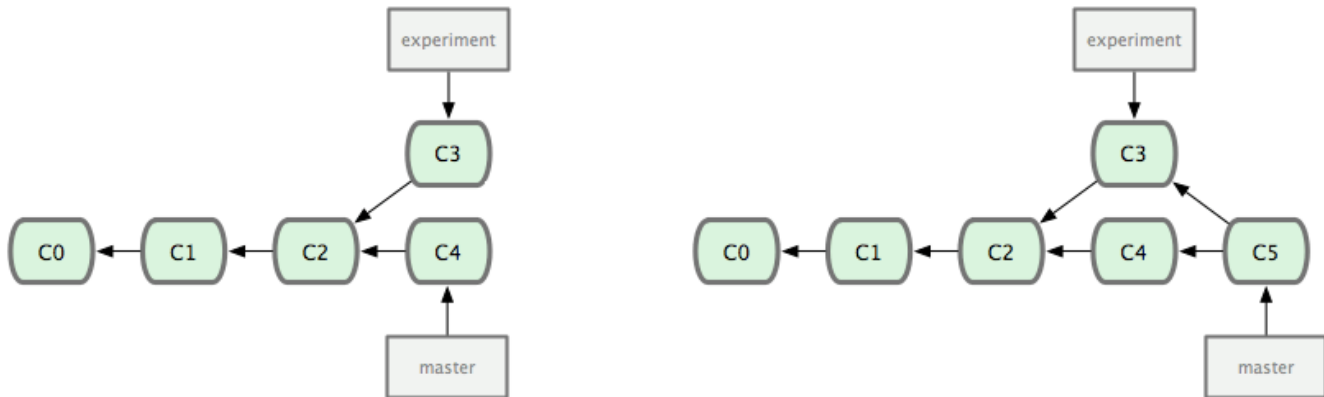
Gestion des branches (2/3) : commandes principales

- Commandes pour gérer les branches
 - **git branch**
 - Liste les branches existantes
 - **git branch f01**
 - Crée la branche « f01 »
 - **git switch f01** ***git checkout f01***
 - Bascule le workspace sur la branche « f01 »
 - **git branch -d f01**
 - Supprime localement la branche « f01 »
 - **git push origin f01**
 - Envoie la branche f01 sur le dépôt distant

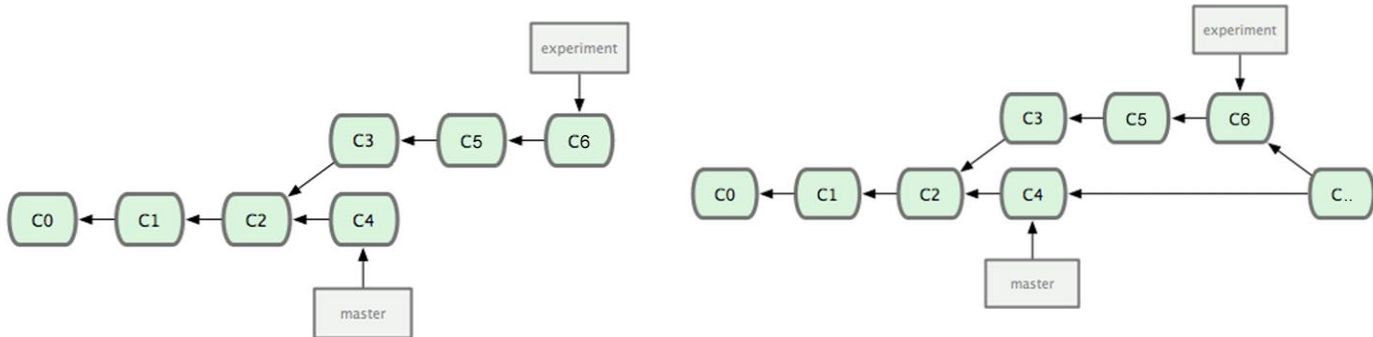
Fusion classique « *fast-forward* » (git merge)



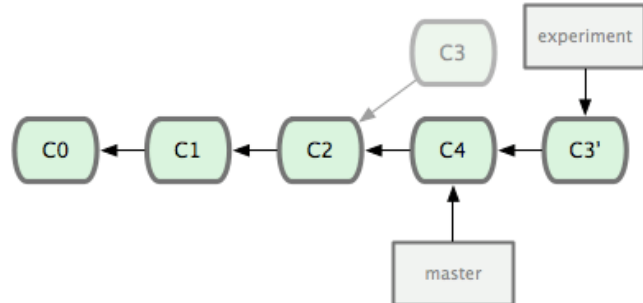
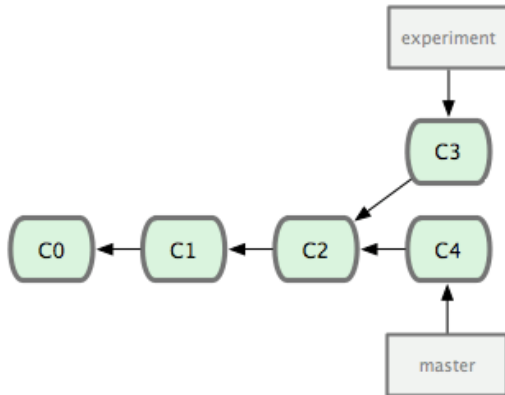
Fusion classique « *recursive / ort* » (git merge)



Fusion classique « *squash* » (`git merge --squash`)



Fusion améliorée (git merge + git rebase)



Exercice

Créer 3 branches depuis « *master* » :

- b1 ; b2 ; b3

Sur chaque branche, créer le fichier adéquat (Bx) et le commiter sur la branche

En local :

- merger *b1* dans *master* (*pas de CF*)
- merger *b3* dans *b2* (*CF*)
- merger *b2* dans *master* via rebase (*pas de CF*)

```
# Création des branches
```

```
git branch b1
```

```
git branch b2
```

```
git branch b3
```

```
# Création des commits initiaux
```

```
git switch b1
```

```
touch B1 && git add B1 && git commit -m "B1"
```

```
git switch b2
```

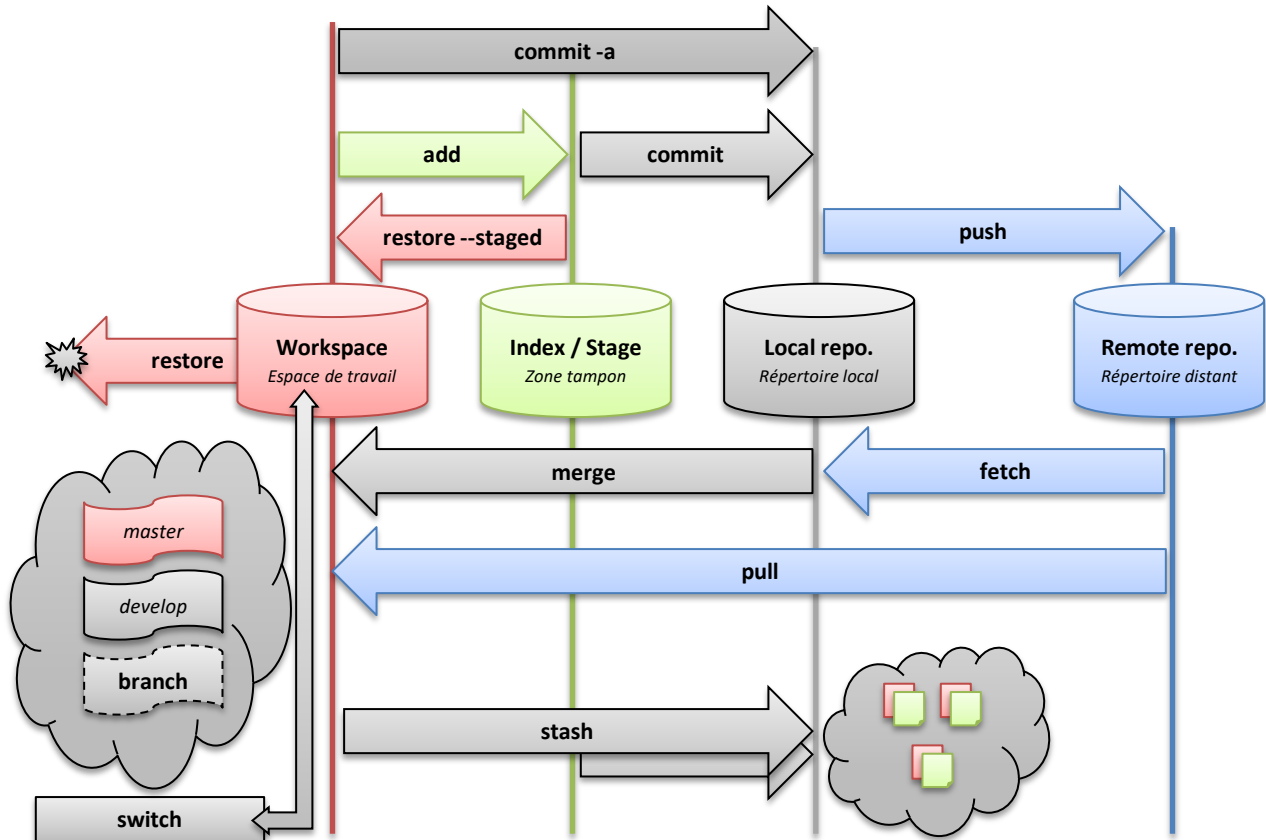
```
touch B2 && git add B2 && git commit -m "B2"
```

```
git switch b3
```

```
touch B3 && git add B3 && git commit -m "B3"
```

CF : Commit de fusion (merge commit)

Schéma des principaux échanges Client-Serveur



COMPLÉMENTS



Créer et appliquer des patches : **git format-patch** / **git am**

- **Méthode 1 : **git diff** et **git apply****
 - Réaliser les modifications voulues sur le « workspace »
 - Générer le patch via « **git diff** »
 - **git diff** > hotfix.patch
 - Appliquer le patch via « **git apply** hotfix.patch »
- **Méthode 2 : **git format-patch** et **git am****
 - Créer une branche « hotfix »
 - Réaliser les modifications voulues et committer
 - Générer le patch via « **git format-patch** *base_branch* »
 - **git format-patch** master
 - Appliquer le patch via « **git am** *****.patch** »

Exercice

Générer un patch qui :

- Modifie le fichier README.md
- Crée le dossier de logs/ avec un fichier .gitkeep à l'intérieur
- Crée le fichier .gitignore à la racine du projet

Tester les deux méthodes

Récupérer un commit spécifique : **git cherry-pick**

- Permet de récupérer un commit spécifique
 - Le commit doit être connu de Git (et accessible)
- **git cherry-pick** *hash*

Exercice

Sur **Github**, créer une branche « *cherry* » et créer un fichier « *correctif* ».
Faire un commit.

En local, utiliser « **git cherry-pick** » pour récupérer le commit
sur la branche « master ».

Ignorer des fichiers : le fichier .gitignore

- Permet d'ignorer certains répertoires et/ou fichiers
 - config/parameters.yaml
 - logs/
 - vendors/
 - ...
- Fichier .gitignore à placer à la racine du dépôt

Exemple de fichier .gitignore

```
# Ignore le fichier
config.yaml

# Ignore le répertoire "logs" et son contenu
/logs/*
# Sauf le fichier .gitkeep
# > Attention, cette ligne doit se trouver après la précédente (/logs)
!/logs/.gitkeep
```

Créer les éléments suivants :

- config.yaml
- logs/.gitkeep

Faire un « **commit/push** »

Créer ensuite quelques fichiers de logs et créer le fichier .gitignore



Github - Les pull requests

- Permet de rendre un « merge » collaboratif
- Outils intégrés au sein de la « pull request » (PR)
 - Espace de discussion
 - Espace de relecture
 - Possibilité d'intégrer des « hooks »
- Mise à jour automatique de la PR en cas de commits
- Différents modes de fusion
 - « Create a merge commit »
 - « Squash and merge »
 - « Rebase and merge »

Créer une branche « pr » en local

- Réaliser des modifications dessus
- Faire un « **commit/push** »

Réaliser une « **pull request** » sur **Github** pour demander la fusion de « *pr* » sur « *master* »



Le remisage : **git stash**

- Permet de mettre de côté (« remiser ») des modifications en cours
 - **git stash** **git stash push -m « msg »**
 - Remise le travail en cours ..en nommant la remise
 - **git stash list**
 - Liste les travaux en cours
 - **git stash show -p**
 - Affiche le « diff » de la remise la plus récente
 - **git stash apply** **git stash apply stash@{2}**
 - Applique la remise la plus récente .. la remise spécifiée
 - **git stash pop**
 - Supprime la remise la plus récente en l'appliquant
 - **git stash drop**
 - Supprime la remise la plus récente sans l'appliquer
 - **git stash branch b01**
 - Applique la remise la plus récente au sein d'une nouvelle branche

Exercice

- Créer une branche « bstash » et modifier le fichier README.md
- Faire un commit
- Retourner sur la branche « master » et modifier le fichier README.md
- Ne pas faire de commit et retourner sur la branche « bstash »
- Utiliser « **git stash** »



Réécrire l'historique : **git rebase -i**

- Permet de réécrire l'historique des N-1 derniers commits (de préférence non propagés)

```
F2000@F2000-PC MINGW64 /d/www/formation (master)
$ git rebase --interactive HEAD~6

pick 31624ff xxxxxxxxxxxxxxxx # Commit n-5
pick f8a6bf1 xxxxxxxxxxxxxxxx # Commit n-4
pick 9bee379 xxxxxxxxxxxxxxxx # Commit n-3
pick 489a458 xxxxxxxxxxxxxxxx # Commit n-2
pick 748923f xxxxxxxxxxxxxxxx # Commit n-1
pick 5aac718 xxxxxxxxxxxxxxxx # Dernier commit
```

- Options possibles :
 - pick, reword, edit, squash, fixup, drop

Exercice

- Utiliser « **git rebase -i** » pour modifier les derniers commits comme suit :

- C6 C6
- C7
- C5 C5'
- C4 C4/C2
- C3
- C2
- C1 C1

```
F2000@F2000-PC MINGW64 /d/www/formation (master|REBASE-i)
$ git rebase --continue
[detached HEAD ba8f2bb] test03--edited
Date: Mon May 6 22:08:08 2019 +0200
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 test03.txt
Stopped at 9bee379980152429a93e326ed14511d53ac77b3e... update
You can amend the commit now, with

    git commit --amend

Once you are satisfied with your changes, run

    git rebase --continue

F2000@F2000-PC MINGW64 /d/www/formation (master|REBASE-i 3/6)
$ git rebase --continue
[detached HEAD 55bd8f9] test04
Date: Mon May 6 22:09:02 2019 +0200
3 files changed, 2 deletions(-)
create mode 100644 test04.txt
create mode 100644 test05.txt
Successfully rebased and updated refs/heads/master.
```

Débogage - Annotations et recherche par dichotomie

- Annoter un fichier
 - « **git blame** *mon_fichier* » permet d'afficher, pour chaque ligne, par qui et quand cela a été modifié
- Identifier un commit « buggé » par dichotomie
 - « **git bisect start** » démarre la recherche
 - « **git bisect bad** *hash* » indique que le commit courant contient le bug à identifier
 - « **git bisect good** *hash* » indique que le commit spécifié NE contient PAS le bug à identifier

La recherche par dichotomie démarre ensuite

- « **git bisect reset** » restaure l'état initial du dépôt

Exercice

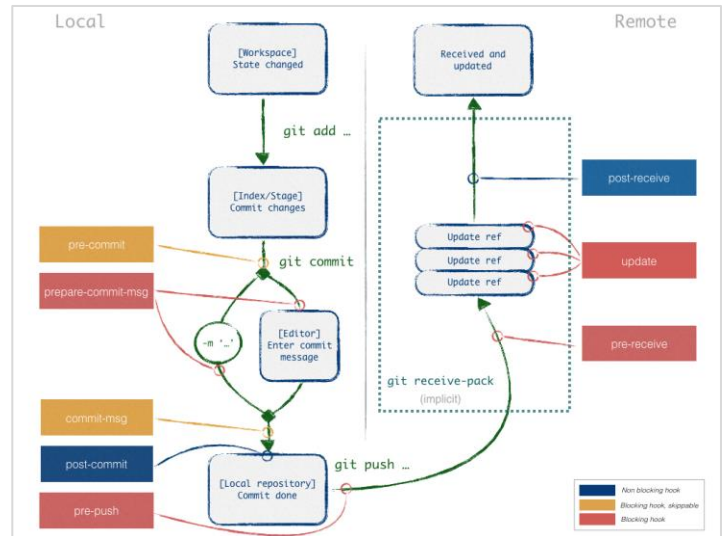
- Tester « **git bisect** » manuellement
- Tester « **git bisect** » via un script

HOOKS ET WORKFLOWS



Les hooks : le dossier .git/hooks

- Permet de lancer des scripts personnalisés à certaines étapes de Git
- Côté « client »
 - « **pre-commit** » : utile pour exécuter des tests ou vérifier des conventions de code.
 - « **prepare-commit-msg** » : permet de personnaliser le message de commit.
 - « **commit-msg** » : permet de valider le message de commit.
 - « **post-commit** » : permet d'effectuer des notifications.
 - « **pre-rebase** » : permet d'empêcher un rebase selon des conditions.
 - Et aussi : « pre-push », « post-rewrite », « post-merge », « post-checkout », ...
- Côté « serveur »
 - « pre-receive », « post-receive »
- Codes erreurs :
0 -> OK >= 1 -> Erreur



<https://delicious-insights.com/fr/articles/git-hooks>

Les hooks : convention de messages

- **Conventional Commits** : spécification ajoutant une signification lisible pour l'homme et pour la machine dans les messages des commits

<type>[optional scope]: <description>

EMPTY LINE

[optional body]

EMPTY LINE

[optional footer(s)]

Message du commit sans corps de texte

```
docs: correct spelling of CHANGELOG
```

Message du commit avec *scope*

```
feat(lang): add polish language
```

Message du commit avec plusieurs paragraphes et plusieurs pieds de page

```
fix: prevent racing of requests
```

```
Introduce a request id and a reference to latest request. Dismiss  
incoming responses other than from latest request.
```

```
Remove timeouts which were used to mitigate the racing issue but are  
obsolete now.
```

```
Reviewed-by: Z  
Refs: #123
```

<https://www.conventionalcommits.org/fr/v1.0.0/#spécification>

Exercice

- Mettre en pratique les hooks suivants :
 - *pre-commit*
 - Vérifier la syntaxe PHP des fichiers « PHP »
 - *pre-commit*
 - Vérifier une liste de mots interdits

Les hooks : les bibliothèques dédiées

Precommit

<https://pre-commit.com>

- Bibliothèque **Python** permettant de facilement pré-configurer des hooks

repos:

- repo: <https://github.com/pre-commit/pre-commit-hooks>
rev: v2.3.0
hooks:
 - id: check-yaml
 - id: end-of-file-fixer
 - id: trailing-whitespace
- repo: <https://github.com/psf/black>
rev: 21.12b0
hooks:
 - id: black

.pre-commit-config.yaml

```
$ pre-commit install
pre-commit installed at /home/asottile/workspace/pytest/.git/hooks/pre-commit
$ git commit -m "Add super awesome feature"
black.....Passed
blacken-docs.....(no files to check)Skipped
Trim Trailing Whitespace.....Passed
Fix End of Files.....Passed
Check Yaml.....(no files to check)Skipped
Debug Statements (Python).....Passed
Flake8.....Passed
Reorder python imports.....Passed
pyupgrade.....Passed
rst "code" is two backticks.....(no files to check)Skipped
rst.....(no files to check)Skipped
changelog filenames.....(no files to check)Skipped
[master 146c6c2c] Add super awesome feature
1 file changed, 1 insertion(+)
```

Overcommit

<https://github.com/sds/overcommit>

- Bibliothèque **Ruby** permettant de facilement pré-configurer des hooks

```
PreCommit:
  RuboCop:
    enabled: true
    command: ['bundle', 'exec', 'rubocop'] # Invoke within Bundler context
```

.overcommit.yml

Les hooks : les bibliothèques dédiées

Husky

<https://github.com/typicode/husky>

- Bibliothèque **JS** permettant de facilement pré-configurer des hooks
 - **commitlint** : respect des conventions sur les messages de commit
<https://github.com/conventional-changelog/commitlint>
 - **commitizen** : *wizard* permettant le respect des conventions sur les messages de commit de manière *user-friendly*
<https://github.com/commitizen/cz-cli>
 - **lint-staged** : permet de lancer des *linters* sur les fichiers modifiés
<https://github.com/okonet/lint-staged>

Exercice

- Mettre en pratique la librairie **Husky**
- Configurer le module **commitlint** afin de *normaliser les messages de commit*
- Configurer le module **commitizen** afin de rendre plus facile la normalisation des messages de commit
- Configurer le module **lint-staged** afin de *rendre nos fichiers plus lisible*

```
package.json
{
  "name": "my-awesome-package",
  "version": "1.0.0"
}

npm install husky --save-dev

package.json
{
  "name": "my-awesome-package",
  "version": "1.0.0",
  "devDependencies": {
    "husky": "^7.0.4"
  },
  "scripts": {
    "prepare": "husky install"
  }
}

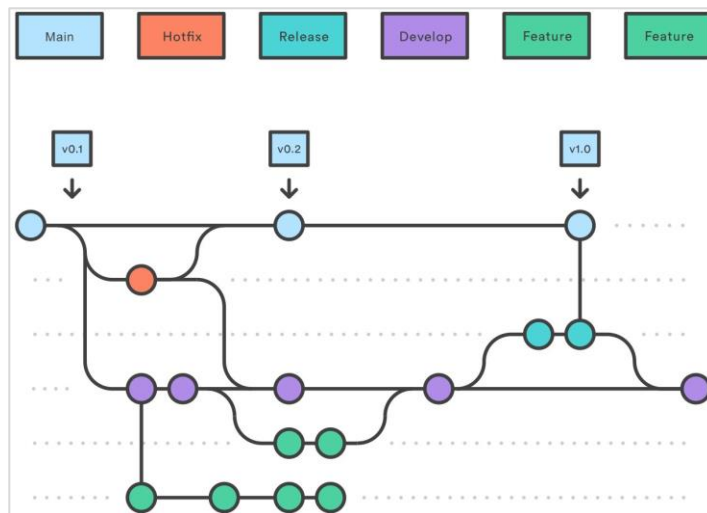
npm run prepare
```

Les workflows

- « **Centralized** » : dépôt centralisé (branche unique) qui sert de point d'entrée unique pour tous les changements apportés au projet.
- « **Git Feature Branch** » : création de branche par fonctionnalité. La branche « main » est isolée et ne contiendra *jamais* de code buggé.
- « **Forking** » : dépôt officiel du projet restreint. Chaque développeur a son propre dépôt serveur pour tester son travail. L'échange d'information se fait au travers de « pull-requests ».

« Git flow » : modèle strict de création de branche

- « **main** » est réservée à la production (*tags*)
- « **develop** » sert de branche d'intégration pour les fonctionnalités
- « **feature** » permet de coder de nouvelles fonctionnalités (« *git feature branch* »)
- « **release** » permet de consolider la future mise en production
- « **hotfix** » permet des patches rapides directement en production



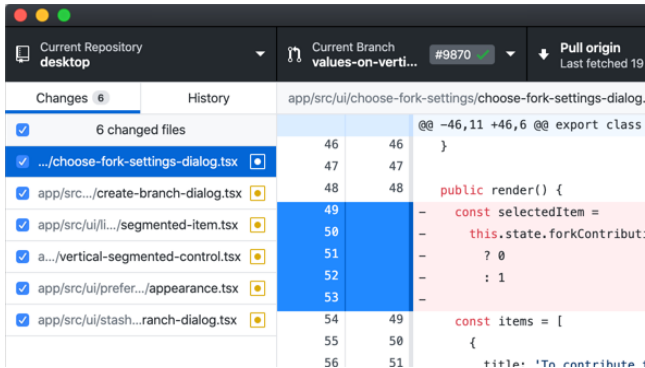
ANNEXES

2

Annexes - Quelques interfaces graphiques pour Git

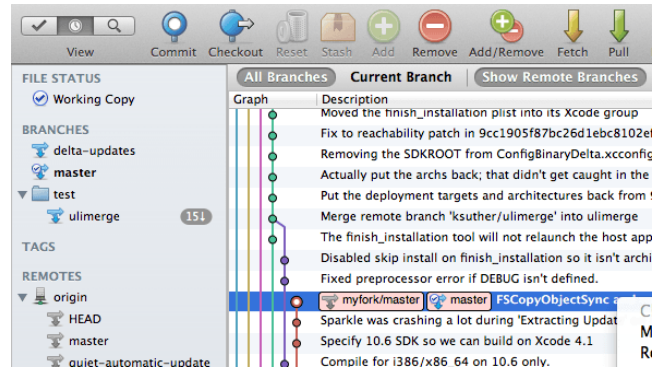
Github Desktop

<https://desktop.github.com>



SourceTree

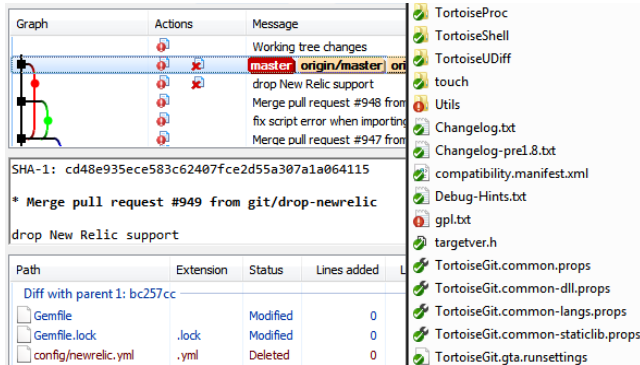
<https://www.sourcetreeapp.com>



Annexes - Quelques interfaces graphiques pour Git

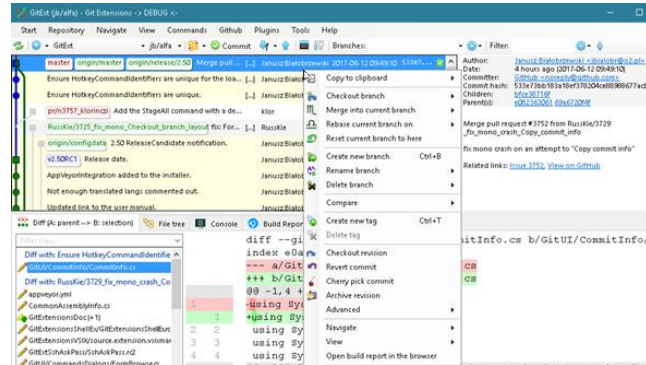
TortoiseGit

<https://tortoisegit.org>



Git Extensions

<https://gitextensions.github.io>



DES QUESTIONS ?

2

LES RUDIMENTS DE GIT



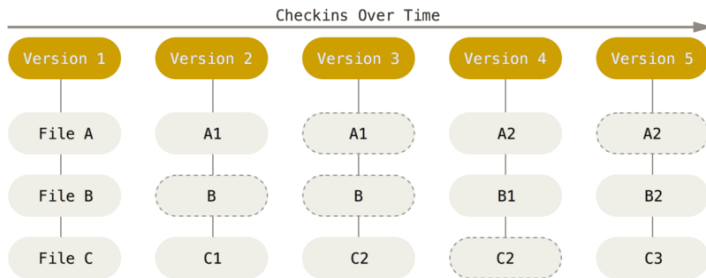
Les rudiments de Git - Le dossier .git

COMMIT_EDITMSG	Dernier message de commit
HEAD	Pointeur sur la branche actuelle
config	Configuration du projet courant
description	Description du projet courant
index	Index binaire des fichiers (<i>git ls-files</i>)
hooks/	Emplacement par défaut des hooks
info/	Fichier “exclude” (<i>usage non recommandé</i>)
logs/	Historique des actions effectuées (<i>git reflog</i>)
objects/	Entrepôt interne des commits
refs/	Références de Git (<i>branches, tags, stashes</i>)

Les rudiments de Git - Gestion de l'historique

Git

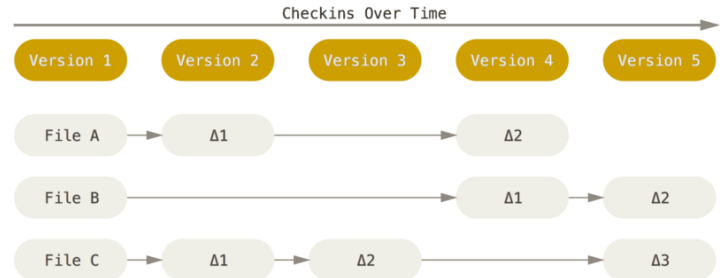
Gestion d'une liste d'instantanés



Autres systèmes

(CVS, Subversion)

Gestion d'une liste de modifications de fichiers



Les rudiments de Git - Gestion des alias

- Modification de la configuration
 - **git config [--global] alias.ci commit**
 - Permet d'effectuer un commit rapide via « git ci »
 - **git config [--global] alias.last "log -1 HEAD"**
 - Permet de visualiser le dernier commit via « git last »
 - **git config [--global] alias.visual "!gitk"**
 - Permet de lancer **gitk** via « git visual »

Les rudiments de Git - Exercice

- Création d'un dépôt
 - Aperçu du dossier .git
- Création d'alias
 - Aperçu du fichier .git/config
- Création d'un premier commit
 - Aperçu du dossier .git/objects
- Création d'un second commit
 - Aperçu du dossier .git/objects

```
# Création du dépôt
# > créer un nouveau dossier avec un fichier lourd SQL
git init
du -hs * .git/*
ls -al .git/**
```

```
# Création d'alias
git config alias.ci commit
git config alias.visual '!gitk'
cat ../git_test/.git/config
```

```
# Création d'un premier commit
git add mon_fichier.sql
du -hs * .git/*
```

```
du -hs .git/objects/*/*
git commit -am "C1"
du -hs .git/objects/*/*

git last
cat .git/refs/heads/master
cat .git/logs/HEAD
cat .git/logs/refs/heads/master
```

```
# Création d'un second commit
# > retirer quelques lignes SQL du fichier
git diff
git commit -am "C2"
du -hs * .git/*
du -hs .git/objects/*/*
```

```
git last
cat .git/refs/heads/master
cat .git/logs/HEAD
cat .git/logs/refs/heads/master
```