



GIT – GÉRER LE VERSIONING

DÉROULEMENT DE LA FORMATION

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

- Jour 2
 - Compléments

2




PRÉSENTATION DE GIT

Jour 1

PRÉSENTATION DE GIT (1/3)

PRÉSENTATION ET UTILITÉ

- Logiciel de gestion de versions <https://git-scm.com>
 - 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

4



PRÉSENTATION DE GIT (2/3)

COMPARAISON AVEC SUBVERSION (SVN)

| | |
|---|--|
| GIT <small>Logiciel de gestion de versions décentralisé</small> | SVN <small>Logiciel de gestion de versions</small> |
| « copie locale » dépot à part entière | « copie locale » copie en lecture du dépôt |
| Permet à ce titre de faire des « commits » locaux | Les commits sont envoyés directement au serveur |

5



PRÉSENTATION DE GIT (3/3)

APERÇU DES FLUX DE TRAVAUX POSSIBLES

| | |
|----------------|--------------------------------|
| « Centralisé » | « Responsable » ayant autorité |
|----------------|--------------------------------|

6



PRISE EN MAIN / COMPRENDRE LES PRINCIPES DE GIT

Jour 1

PRISE EN MAIN (1/2)

INSTALLATION ET CONFIGURATION

- o Installation sous Windows <https://git-scm.com>

- Git Bash émulateur de console Unix
- Git GUI interface graphique
- Intégration automatique dans Windows



- o Configuration

- git config --global user.name "votre_pseudo"
- git config --global user.email moi@email.com
- git config --list

PRISE EN MAIN (2/2)

COMMANDES PRINCIPALES

- o Commandes pour démarrer
 - git init Création du dépôt dans le répertoire courant
 - git init mon-depot 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

```
F20000F2000-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)
```

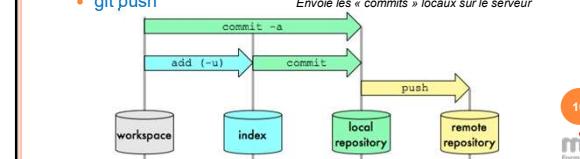


PRISE EN MAIN (2/2)

COMMANDES PRINCIPALES

- o Commandes pour envoyer des modifications

- git add ... Ajoute des fichiers dans l'index local
- git reset HEAD ... Retire des fichiers de l'index local
- git rm ... 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



EXERCICE

Créer un premier dépôt en local.

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

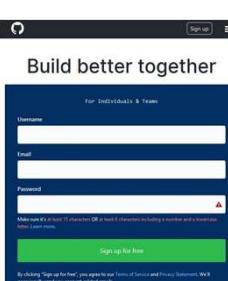
```
F20000F2000-PC MINGW64 /d/www/formation (master)
$ touch hello.txt
$ echo "Hello world!" > hello.txt
$ git add hello.txt
$ git commit -m "Initial commit"
Your branch is up-to-date with 'origin/master'.
  (use "git push" to publish your changes)
nothing added to commit but untracked files present (use "git add" to track)
```

Ajouter et intégrer ce fichier dans un « commit ».

```
F20000F2000-PC MINGW64 /d/www/formation (master)
$ git commit -m "Test commit"
nothing to commit, working directory clean
$ git add hello.txt
$ git commit -m "Adding file"
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your changes)
nothing added to commit but untracked files present (use "git add" to track)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your changes)
nothing to commit, working directory clean
```



GITHUB



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

Fonctionnalités annexes :
- « Bugtracker »
- « Wiki »



EXERCICE

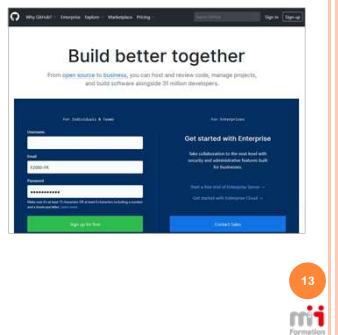
Créer un dépôt distant grâce à Github.

Envoyer le commit précédent sur le dépôt distant.

Astuce :

`git remote add ...`

Pour ajouter un dépôt distant



PRISE EN MAIN (2/2)

COMMANDES PRINCIPALES

o Commandes pour recevoir des modifications

- `git pull ...`

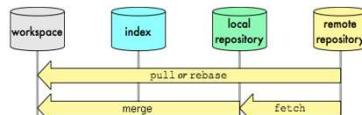
Récupère des modifications depuis le serveur et les applique sur le répertoire de travail

- `git fetch ...`

Récupère des modifications depuis le serveur

- `git merge`

Applique les modifications sur le répertoire de travail

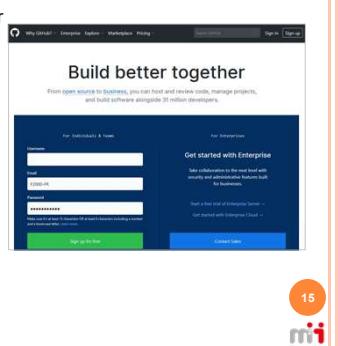


EXERCICE

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

Essayer les commandes suivantes :

- `git fetch`
- `git status`
- `git merge`
- `git status`
- `git push`



TRAVAILLER EN ÉQUIPE

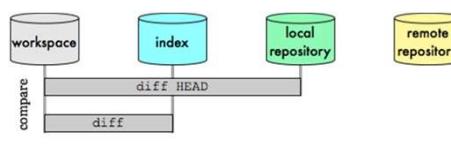
Jour 1 / Jour 2

VOIR LES DIFFÉRENCES EN LOCAL

GIT DIFF / GIT DIFF HEAD

o Lorsque l'on modifie plusieurs fichiers, il peut être utile de réafficher les modifications effectuées

- `git diff`
 - affiche les modifications par rapport à l'index
- `git diff HEAD` (ou `git diff --cached`)
 - affiche les modifications par rapport au dépôt local



VOIR L'HISTORIQUE DES CHANGEMENTS

GIT LOG

o 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 » 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**

The screenshot shows a GitHub repository interface. At the top, there's a navigation bar with links for Code, Issues, Pull requests, Projects, Wiki, Insights, and Settings. Below the navigation bar, a message says "No description, website, or topics provided." A "Manage topics" button is available. The main area displays a merge pull request from user 'johndoe' to 'main'. The pull request has 8 commits, 1 branch, and 4 releases. It has 1 contributor. The status bar at the bottom indicates "Branch: master > New pull request".



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
# Project Git

Ceci est une formation M2i.
<<<MERGE
# Fubar;
<<<COMMIT

foobar is a Python library for dealing with word pluralization.
```



EXERCICE

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

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

Effectuer les commandes suivantes :

- git fetch
 - git status
 - git merge

Corriger le conflit

The screenshot shows the GitHub homepage. At the top, there's a navigation bar with links for 'Why GitHub', 'Enterprise', 'Explore', 'Marketplace', and 'Pricing'. Below the navigation, a large banner features the text 'Build better together' in a bold, white, sans-serif font. Underneath the banner, a subtext reads 'From open source to business, you can host and review code, manage projects, and build software alongside 31 million developers.' To the right of the banner, there's a 'Get started with GitHub' button. Further down the page, there's a section titled 'Get started with Enterprise' with a subtext about taking collaboration to the next level with enterprise features like 'Custom domains' and 'Enterprise DevOps'. At the bottom, there's a 'Sign up for free' button and a 'Contact Sales' button.



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 « `--hard` » efface définitivement les modifications
 - **Désindexer un fichier**
 - La commande « `git reset HEAD ...` » ou « `git restore --staged` » permet de désindexer tout l'index courant (ou un fichier spécifié)
 - **Réinitialiser un fichier modifié**
 - La commande « `git checkout` » ou « `git restore` » 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)

25
mi Formation

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 [...]`

26
mi Formation

GESTION DES BRANCHES

Jour 1

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 checkout f01` `git switch 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

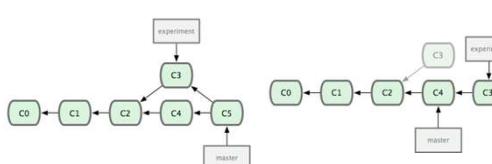
29
mi Formation

GESTION DES BRANCHES (3/3)

FUSIONNER DES BRANCHES : MERGE ET REBASE

git merge

git rebase + merge



30
mi Formation

EXERCICE

Créer 3 branches depuis le « master » :

- b01 ; b02 ; b03

Sur chaque branche, créer le fichier adéquat (test_b0x.txt) et le commiter sur la branche

En local :

- merger b01 dans master (*pas de CF*)
- merger b03 dans b02 (*CF*)
- merger b02 dans master via rebase (*pas de CF*)



31

COMPLÉMENTS

Jour 2

CRÉER ET APPLIQUER DES PATCHS

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
 - Générer le patch via « `git format-patch base_branch` »
 - `git format-patch master`
 - Appliquer le patch via « `git am ***.patch` »

34



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



35

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`

36



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 ».

37
mi Formation

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 »

38
mi Formation

EXERCICE

Créer une branche « pr » en local



39
mi Formation

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

Réaliser une « **pull request** » sur **Github** pour demander la fusion de « pr » sur « 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/*
# Sauve le fichier .gitkeep
# > Attention, cette ligne doit se trouver après la précédente (/logs)
!logs/.gitkeep
```

40
mi Formation

EXERCICE

Créer les éléments suivants :

- config.yaml
- logs/.gitkeep

Faire un « commit/push »

41
mi Formation

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

LE REMISAGE

GIT STASH

- Permet de mettre de côté (« remiser ») des modifications en cours
 - `git stash`
 - Remise le travail en cours
 - `git stash list`
 - Liste les travaux en cours
 - `git stash show -p`
 - Affiche le « diff » de la remise la plus récente
 - `git stash apply`
 - 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

42
mi Formation

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](#) »

43
 mi Formation

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, exec, drop

44
mi Formation

EXERCICE

- Utiliser « `git rebase -i` » pour modifier les derniers commits comme suit :
 - C6 C6
 - C7 C7'
 - C5 C5'
 - C4 C4
 - C3 C3/C4
 - C2 C2
 - C1 C1

45

DÉBOGUAGE

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 « bugué » par dichotomie
 - « `git bisect start` » démarre la recherche
 - « `git bisect bad` » 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

46

EXERCICE

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

47

LES HOOKS

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` »

48

EXERCICE

- Mettre en pratique les hooks suivants :
 - pre-commit
 - Vérifier l'absence de lignes blanches en fin de fichiers

49
mi
Formations

DES QUESTIONS ?

Jour 2