



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



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

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

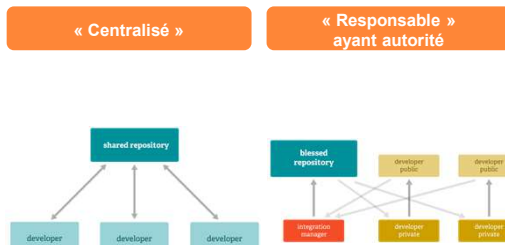
SVN
Logiciel de gestion de versions

« 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



PRISE EN MAIN / COMPRENDRE LES PRINCIPES DE GIT

Jour 1

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`

8



PRISE EN MAIN (2/2)

COMMANDES PRINCIPALES

- 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*

```
C:\0000F2000-PC\WINGW64 /d/www/Formation (master)
$ git status
On branch master
Initial commit
nothing to commit (create/copy files and use "git add" to track)
```

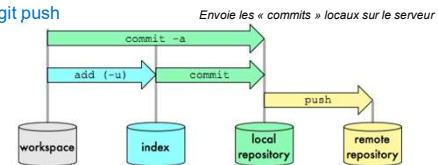
9



PRISE EN MAIN (2/2)

COMMANDES PRINCIPALES

- 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*



10



EXERCICE

Créer un premier dépôt en local.

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

Ajouter et intégrer ce fichier dans un « commit ».

```
C:\0000F2000-PC\WINGW64 /d/www/Formation (master)
$ touch hello.txt
C:\0000F2000-PC\WINGW64 /d/www/Formation (master)
$ git add hello.txt
C:\0000F2000-PC\WINGW64 /d/www/Formation (master)
$ git commit -m "first commit"
[master (root-commit) 1b1b1b1] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 hello.txt
C:\0000F2000-PC\WINGW64 /d/www/Formation (master)
$ git status
On branch master
nothing to commit, working directory clean
```

11



GITHUB



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

Fonctionnalités annexes :

- « Bugtracker »
- « Wiki »

12



EXERCICE

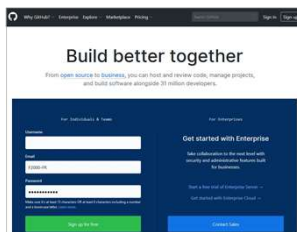
Créer un dépôt distant grâce à Github.

Envoyer le commit précédent sur le dépôt distant.

Astuce :

`git remote add ...`

Pour ajouter un dépôt distant



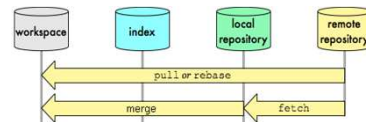
13

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



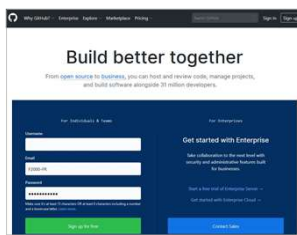
14

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`



15

TRAVAILLER EN ÉQUIPE

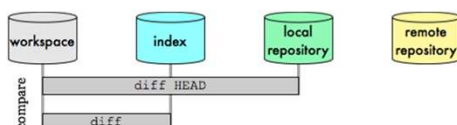
Jour 1 / Jour2

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



17

VOIR L'HISTORIQUE DES CHANGEMENTS

GIT LOG

o Après de nombreux « commits », il peut être intéressant d'afficher l'historique des modifications

- `git log`
 - o affiche les différents commits effectués sur le dépôt
- `git log -p`
 - o affiche le détail des différents commits effectués sur le dépôt
- `git show hash`
 - o Affiche le détail d'un commit spécifique grâce à son « hash »

18

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

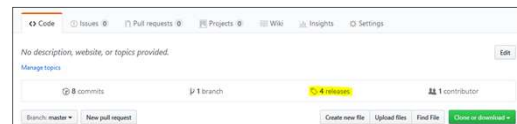
19



EXERCICE

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

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



20



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

```
P2000@P2000-PC MINOW64 /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.

P2000@P2000-PC MINOW64 /d/www/formation (master/MERGING)
$ cat README.md
#<div data-bbox="142 565 247 585" data-label="Text">

```
##<div data-bbox="142 585 179 599" data-label="Text">

```
# Foobar
```


```


```

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

21



EXERCICE

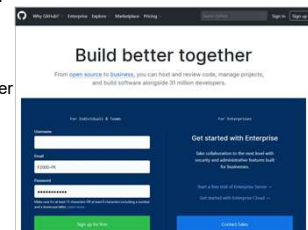
Sur Github, modifier le fichier README.md et créer un commit.

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



22



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.

23



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` »

24



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



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)
 - `git revert [...]`

26



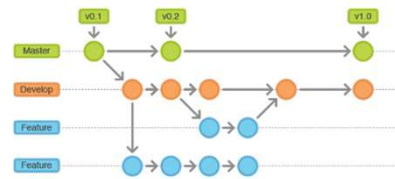
GESTION DES BRANCHES

Jour 1

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



28



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

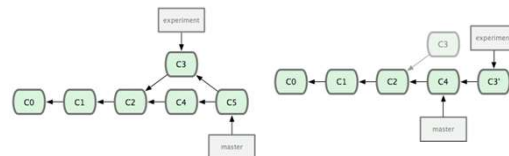


GESTION DES BRANCHES (3/3)

FUSIONNER DES BRANCHES : MERGE ET REBASE

`git merge`

`git rebase + merge`



30



EXERCICE

Créer 3 branches depuis le « master » :

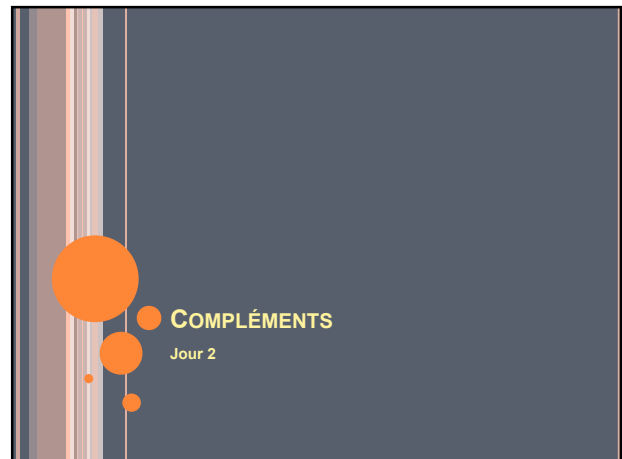
- b01 ; b02 ; b03

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

En local :

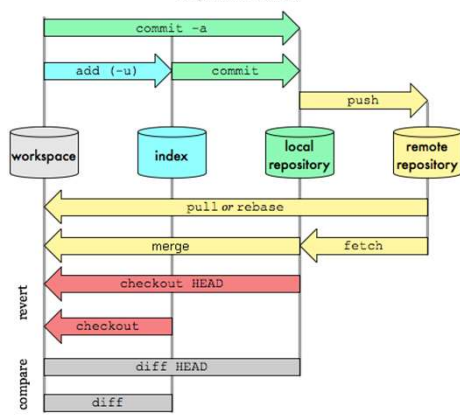
- merger b01 dans master (*pas de CF*)
- merger b03 dans b02 (*CF*)
- merger b02 dans master *via rebase* (*pas de CF*)

31



Git Data Transport Commands

<http://cateele.com>



33



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



GITHUB

LES PULL REQUESTS

- o Permet de rendre un « merge » collaboratif
- o 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
- o Différents modes de fusion
 - « Create a merge commit »
 - « Squash and merge »
 - « Rebase and merge »

38



EXERCICE

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 »



39



IGNORER DES FICHIERS

LE FICHIER .GITIGNORE

- o Permet d'ignorer certains répertoires et/ou fichiers
 - config/parameters.yaml
 - logs/
 - vendors/
 - ...
- o 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
```

40



EXERCICE

Créer les éléments suivants :

- config.yaml
- logs/.gitkeep

Faire un « commit/push »

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

41



LE REMISAGE

GIT STASH

- o Permet de mettre de côté (« remiser ») des modifications en cours
 - `git stash` `git stash save « 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

42



- 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` »



GIT REBASE -I

- ```
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 |

- 

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

- ```
git release --continue
[detached HEAD ba7703b] test03--edited
Date: Mon May 2 21:09:00 2010 +0200
files changed:
  create mode 100644 test03.txt
Create mode 100644 test03.txt
git commit -m "test03"
[detached HEAD 411d34c] 411d34c3ac7b67... update
You can amend the commit now, with

    git commit --amend

once you are satisfied with your changes, run

    git release --continue

$ cd /usr/src/linux
$ git release --continue
[detached HEAD 93d9e97] test04
Date: Mon May 2 21:09:00 2010 +0200
files changed:
  create mode 100644 test04.txt
Create mode 100644 test04.txt
git commit -m "test04"
[detached HEAD 1064a7e] 1064a7e2... update
Successfully rebased and updated refs/heads/master.
```



ANNOTATIONS ET RECHERCHE PAR DICHOTOMIE

- La recherche par dichotomie démarre ensuite

- 

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



DOSSIER .GIT/HOOKS

-

EXERCICE

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

49



DES QUESTIONS ?

Jour 2