Outil de versioning collaboratif de projet

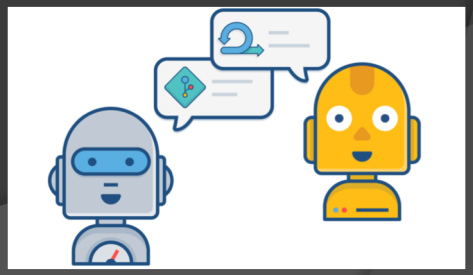
GIT

A quoi ça sert ?

1. Permet de créer plusieurs versions d'un projet

2. Permet de sauver à distance un projet (cloud)

3. Permet de travailler à plusieurs sur un même projet



1) La notion de version

- Permet de répondre à
  - Qu'est ce qui a été modifié ?
  - Pourquoi ?
  - Quand ?
  - Par qui ?
- Commit : révision (nouvelle version) de un ou plusieurs fichier(s)
- Permet de revenir à la version précédente

Les gestionnaires de version

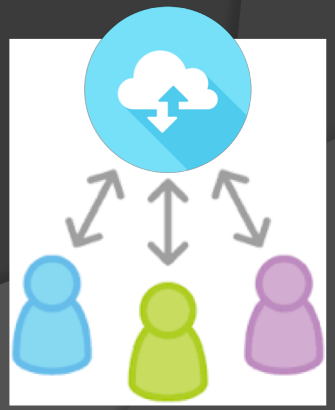
- CVS
- SVN
- Mercurial
- Bazaar
- GIT
  - Très puissant et récent, il a été créé par *Linus Torvalds*. Il se distingue par sa rapidité et sa gestion des branches qui permettent de développer en parallèle de nouvelles fonctionnalités.

Qu’est ce qu’un dépôt (local/distant)

=> Un dossier ou est stocké le projet

- Local: le code est versionné sur le disque (mode pas d’amis)

- Distant: dépose d’une version à distance
  - Pour sauvegarder Cloud
  - Pour partager ses sources
  - Pour que les autres bossent (les amis)
  - ex : Github, BittBucket



Les outils GIT (1/2)

- Ligne de commande : git
- Installer git for windows
  - https://gitforwindows.org/
- Directement intégrés dans l'IDE

Création d'un repository local

- Endroit ou déposer les fichiers

- Comment le créer
  - Lignes de commande : git init

  - Ou Github Desktop
  - Ou sur Github.com puis le cloner

Kit de survie LINUX

- pwd : ou suis-je ?
- ls : listons le dossier courant
- mkdir <dossier> : créons un dossier
- cd <dossier> : changeons de dossier

lignes de commandes : git
Porcelaine et plomberie

- git init crée un repository local

- git add <fichier>ajoute le contenu du fichier à l'index pour le prochain commit
- git add –u ajoute tous les fichiers modifiés

- git commit –m "message" enregistre dans le dépôt **local** les fichiers à l'index.
- git commit –a –m "message" fait le add et le commit

- git status affiche les fichiers/dossiers qui ont été modifiés depuis le dernier commit
- git diff montre les différences
- git log liste l'historique des commits
- git rm supprime un fichier de git et de l'index

- git reset –hard efface tous les changements des fichiers et retour au dernier commit
- git checkout -- <fichier> revenir à la dernière version du fichier (destruction nouvelles modifs)

- git help permet d'afficher l'aide interne de git.
- Pour avoir des détails sur une commande particulière : git help <commande>

Création repository local

Création du projet / projet existant

```
mkdir monProj
cd monProj
```

Init du projet et vérif

```
git init
git status
```

Version : add+commit

- Version : «ensemble de changements qui permet soit d'ajouter une fonctionnalité, soit de régler un bug»
- add : création de l'index des fichiers candidats à nouvelle version
- commit : création de la nouvelle version

création 1ère version

... éditer index.html ...

- add + commit

```
git add index.html
git commit –m “Création index.html”

git status
```

Création 2nde version

```
... modification index.html ...
Regarder les différences avec la
dernière version
git status
git diff

création 2nde version
git add index.html
git commit –m “qqs modifs cosmetiques”
```

One of those day

```
Création du projet git (1 seule fois lors de la création du projet)
mkdir monProj
cd monProj
git init
git status

Première édition du html + création 1ère version
... éditer index.html ...
git status
git add index.html
git commit –m “Premier Html”
git status

... modification index.html ...
Regarder les différences avec la dernière version
git status
git diff

création 2nde version
git status
git add index.html
git commit –m “qqs modifs cosmetiques”
git status

... Création main.css ...
git status
git add css
git commit –m “Premier Css”
git status

Regarder l’historique des commits
git log

faire tous les add et les commits d’un coup
git status
git add –u
git commit –m “ qqs corrections de bugs”
```

Fork : création d'une branche

- Branche principale (*master) : branche de référence de l'avancement du projet
- Plusieurs branches : travail en parallèle
- Ne pas polluer la branche principale
  - une branche par nouvelle fonctionnalité
  - une branche par personne
    - plusieurs branche/personne si nécessaire

création d'une branche

Création branche pour la page contacts

```
git branch contacts
```

Verification

```
git branch
```

Changement branche

```
git checkout contacts
```

Travail dans une branche

- Créer contacts.html
- Ajouter dans index.html

```
<a href="contacts.html"></a>
```

- Faire les add et commit

```
git add contacts.html index.html
git commit –m “ajout du contact”
```

Travail dans une branche

Retour au Master

```
git checkout master
```

Regarder le code de index.html

Modifier le code de index.html…

```
git add index.html
git commit –m “ajout h3”
```

Intégrer le travail de la branche dans le master

Etre dans le master

```
git checkout master
```

(Optionnel)vérifier les différences

```
git diff master..contacts
```

Faire un merge de branche

```
git merge contacts
```

Suppression de la branche

Bonne pratique

```
git branch –d contacts
```

Les conflits

- Quand deux branches modifient le même fichier, il y a conflit , le commit est bloqué
- Il faut résoudre le conflit en remplaçant tout le bloc marqué par

  <<<< head

  …

  >>> branch

  Puis [Click droit] “Mark as resolved”

Résolutions des conflits « Eventuels » liés au Merge

```
... Editer les fichiers à prbs
… Résoudre les prbs
… Faire les add commit du master
git add index.html
git commit –m “Conflits résolus”
```

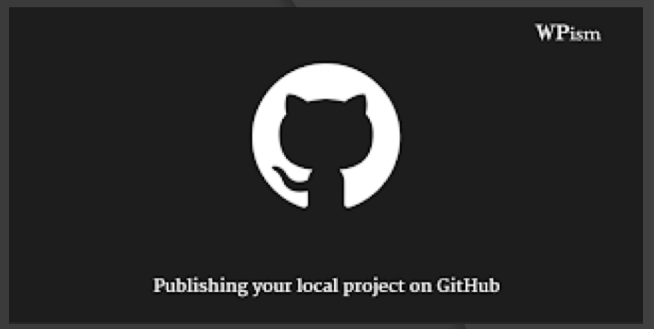
Exercice Fork

- *Créer une branche apropos*
- *Créer une page apropos.html avec son nom, son mail, sa photo.*
- *Faire*
  - *1er commit avant l'ajout de la photo*
  - *Second commit après*
- *Créer un conflit entre le master et apropos*
  - *Changer le titre de la page dans le master*
  - *Merger dans le master*

Les outils (2/3)

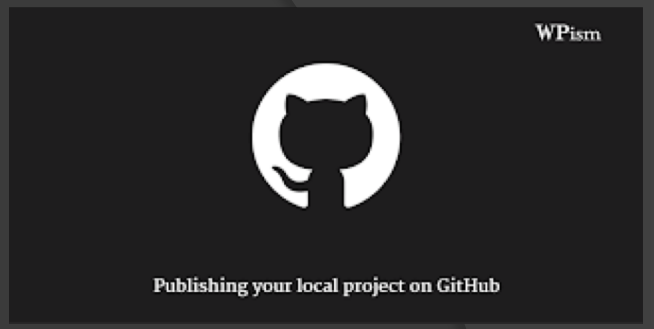
- Interface graphique
  - Github desktop
    - download sur desktop.github.com

Les outils (2/2)



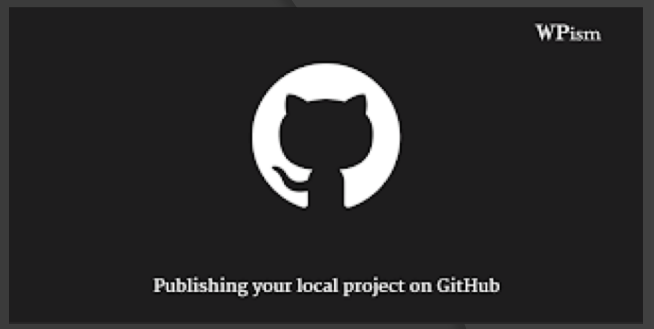
- ~~Ligne de commande : git~~
- Interface graphique
  - Github desktop
    - download sur desktop.github.com

GitHub desktop



- Interface graphique de git
- Comment faire plus simple et plus beau
- (Mais en maitrisant moins ce qui se passe)

Create Repository GitHub desktop

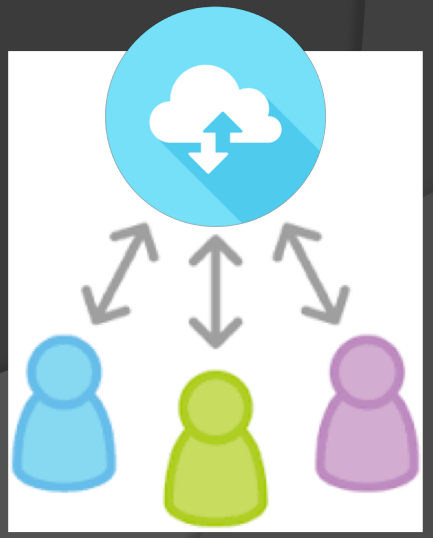


- [Local] New Repository / +

- Exercice 1
  - *Faire un nouveau repository*
  - *Y ajouter un fichier Html*
    - *Contenant votre titre et votre photo (par ex)*
  - *Faire des commit*
  - *Créer une branche*
  - *La merger*

La notion de dépôt local/distant

- Local: le code est versionné en local
- Distant: dépose d'une version à distance
  - Pour sauvegarder: Cloud
  - Pour que les autres la récupère
  - ex : Github, BittBucket



Github.com

- Webservice de création de dépots public (gratuits) ou privés (payants)
- Permet la consultation, clonage, download .zip
- Le service propose également des **fonctionnalités sociales** (*profil utilisateur, statistiques, suivi d'utilisateurs et de projets…*) et d'interconnexion avec d'autres outils.

Création d'un compte

- Github.com
- Sign Up

- Exercice
  - Créer un compte sur gitHub.com

Le publish-push/sync

- Publier votre Version : sur github.com

Clone Repository GitHub desktop

- Permet de récupérer un projet
  - Github.com -> clone local
- Github Desktop: File Clone / + Clone
- Explore Github
- Quand on veut travailler dessus, on commence par faire un fork, (à la fin un pull request)
- Exercice
  - *Cloner le repository du voisin*
  - *Rajouter sa trombine dans sa page*

Travail à plusieurs

- Le propriétaire du projet doit inviter des contributeurs
  - Leurs donner les droits qu'ils souhaitent
    - [Write] leur permet de faire des push
- (Autre solution plus compliquée faire un Team)
- Exercice
  - *Aller dans settings de GitHub pour ajouter un contributeur*

Méthode de travail 1 : simple

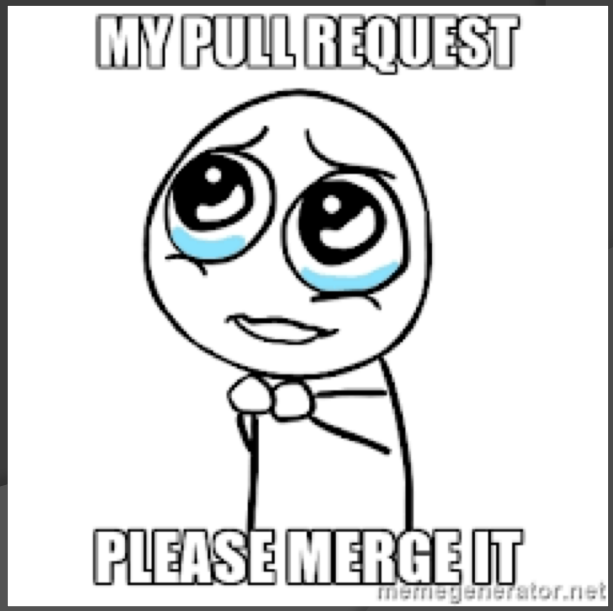
- Chaque collaborateur fait régulièrement des Pull pour récupérer les modifs.
- Chaque collaborateur fait régulièrement des Push de ses modifs.

Méthode de travail 2 : évoluée

- Chaque utilisateur fait un Fork pour avoir sa version. Il fait ses Commit et ses Push dans sa branche
- Quand il considère avoir terminé sa branche, il la fusionne avec la principale : pull-request

Pull-request

- Demande d'une branche au dépôt principal
- Pour accepter cette demande par tous, il faut faire un merge-request



Le Fork : création d'une branche

- Par défaut, un projet créé sur Github ne peut recevoir des commits que de son créateur, à moins que celui-ci valide une **pull-request**.
  Autrement, il peut également ajouter d'autres utilisateurs auxquels il donnera un accès **Push** (collaborateurs).

lignes de commandes distantes

- git pull récupère les changements qui auraient été pushés par d'autres utilisateurs, il fonctionne comme git fetch à la différence qu'il fait un merge en plus

- git push envoie les derniers commits effectués vers ce dépôt

- En cas de conflit, tenter de résoudre manuellement le conflit en ouvrant le fichier en question dans votre éditeur, puis une fois le fichier corrigé, vous effectuer git add <fichier> pour le marquer comme résolu et le réintroduire dans l'index.

Recap

///// Installation :

git init

Créer un nouveau dépôt dans un dossier

git clone x

Télécharger et installer un dépôt déjà existant dans un dossier (x = url du dépôt distant)

///// Configurations :

git config --global user.name "X"

Assigner X comme nom qui apparaîtra dans les logs

git config --global user.email "X »

Assigner X comme mail qui apparaîtra dans les logs

Recap

git status
Donne le détail des fichiers non indexés et non validés dans le dépôt (en rouge = non indexés, en vert = indexé mais non validé)
git add X
Indexe le fichier spécifié X (git add * ajoutera d'un coup tous les fichiers non indexés)
git diff
Liste les différences entre les fichiers du dépôts et les fichiers indexés
git commit -m "x"
Crée un commit ajoutant au dépôt tous les fichiers indexés (fichiers en vert dans le git status).
git push
Envoi les derniers commits créés sur le dépôt distant
git rm
Supprime un fichier (-f pour supprimer ce fichier aussi dans le dépôt, --cached pour supprimer le fichier uniquement dans l'index)
git mv "X" "Y »
Déplace et/ou renomme un fichier X en Y
git log
Affiche les logs des commits réalisés (-x pour afficher les x derniers commits, -p pour voir les détails, --oneline pour compacter en une ligne)
gitk
Interface graphique de l'historique du dépôt
git reset HEAD X
Enleve le fichier X de l'index (passe le fichier du vert au rouge dans git status)
git checkout -- X
Annule tous les changements réalisés sur le fichier X dans l'espace de travail depuis le dernier commit
git fetch origin
Récupère les derniers commits réalisés sur le dépôt distant pour fusionner avec le dépôt local
git pull
Permet de mettre à jour le dépôt local avec le dépôt distant

Recap : Travailler sur le projet:

1 - git pull (permet de récupérer la dernière version du projet)

// Après avoir finit le travail sur le projet:

2 - git status
(liste les changements réalisés sur l'espace de travail, à ce stade tout doit être rouge)
3 - git add *
(ajoute tous les fichiers modifiés à l'espace de validation)
4 - git status
(à ce stade tout doit être vert, sinon recommencer depuis l'étape 2)
5 - git commit -m "Description de la mise à jour" (valide les modifications des fichiers ajoutés)
6 - git status
(si tout est blanc (ni de rouge, ni de vert), alors passer à la suite, sinon refaire depuis l'étape 2)
7 - git push
(envoi les modifications validées au serveur.)

// Si l'étape 7 donne une erreur de type "ERROR Rejected", cela veut dire que quelqu'un d'autre a effectué d'autres mofidications sur le projet. Dans ce cas, continuer à l'étape 8.
8 - git fetch origin
(permet de récupérer les dernières modifications du projet)
9 - git merge origin/master -m "fusion"
(fusionne les modifications récupérés précédement avec nos modifications.)
// SI ERREUR de type "conflict", régler les fichiers concernés et refaire depuis l'étape 2
10 - git push
(envoi les modifications validées au serveur.)
// Si il y a encore une erreur, quelqu'un a encore pu modifier entre temps le projet sur le serveur, donc recommencer à l'étape 8

Entrainement :
Tp1 : ce que vous devez savoir faire

- Créer un repository
- Y ajouter des fichiers html/css/js
- Faire des modifs
- Faire des commit

- `git init/ git add/ git comit –m/ git status/ git show / git diff`
- Vérifiez/modifier avec github desktop

Tp2 : ce que vous devez savoir faire

- Récupérer un projet existant sur un repository / créer un repository
- Faire des modifs
- Push aux autres
- Récupérer les modifs des autres

Tp3 : Un TP d'équipe par 3

- Member 1 :
  - Crée le repository
  - S'occupe du html
- Member 2
  - Fait les images et le Css
- Member 3
  - Fait le javascript

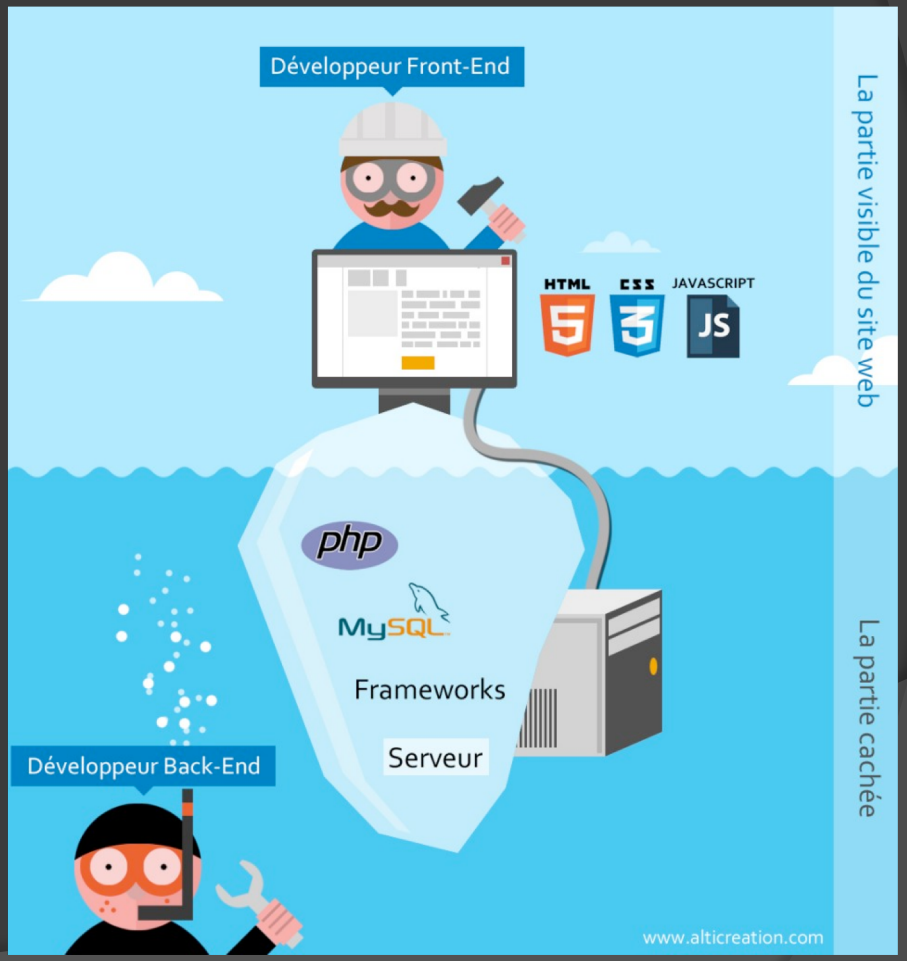
TpFinal : Un TP d'équipe par 2

- L'objectif est de travailler à 2 sur un projet en utilisant git
- Dans chaque projet il y aura 1 chef de projet et 1 collaborateur. Le collaborateur sera celui placé à gauche du chef de projet.
- Le chef de projet crée une page .html et .css
- Le collaborateur clone le projet, se fait une branche, crée sa page monNom.html et modifie index.html pour y intégrer un lien vers sa page
- Pour le fun : faites des conflits !!!

Refaire :

http://laurent-freund.fr/wf3/cours/17_intro/

- Avec répartition
- 2 développeurs



Les issues

- Permet de commenter, de donner des infos à des personnes du projet lors des commit et des pull-request

Le wiki

- Description du projet

Divers

- Un tuto

  http://learngitbranching.js.org/