Git : documentation

Gérer un dépôt

Qu'est-ce qu'un dépôt ?

Le dépôt peut-être :

- un répertoire disposant déjà d'un projet
- un répertoire vide dédié à un projet futur (...ou proche)
- le clone d'un dépôt existant `git clone`

A lire ... Démarrer un dépôt git

git init

`git init` est utile au démarrage d'un dépôts sans dépôt distant au préalable.

Pour initialiser un dépôt sur votre ordinateur, il faut :

1. se positionner sur le répertoire de dépôt puis clic-droit + `Git Bash`
2. taper la commande

```
$ git init
```

Cela fait, un sous-répertoire `.git` est créé. Il contient tous les fichiers nécessaires à la gestion du dépôt.

Exemple

Mise en place d'un dépôt pour le projet `plum.mvc`:

1. Créer un répertoire : `c:/git/`
2. Créer un dépôt : `c:/git/plum.mvc/`
3. Se positionner sur le répertoire `c:/git/plum.mvc`[à l'aide de l'explorateur windows]
4. Cli-droit dans la fenêtre de l'explorateur windows : choisir `Git Bash`
5. Initialiser le répertoire : `git init` :[**git** ajoute le répertoire caché `.git`]
6. Copier le contenu du projet `plum.mvc` se trouvant sur un serveur Web
7. Indexer les fichiers du projet : `git add .` [commande équivalente `git add –all`]
8. Validation de l'indexage : `git commit -m 'version 0'`
9. Poser un tag sur le dernier committ : `git tag -a version.0 -m 'version.0 exemple non terminée'`

Voir http://git-scm.com/book/fr/Les-bases-de-Git-D%C3%A9marrer-un-d%C3%A9p%C3%B4t-Git

Dépôt distant

La gestion d'un dépôt distant nécessite au préalable

- de connaitre l'adresse du dépôt distant `git`. Par exemple `git@37.187.66.61:/plummvc.git` ;
- de disposer de droit de lecture et d'écriture ;
- de disposer d'une clé SSH liée à un compte. Dans notre exemple le compte porte le nom de `git`

L'administrateur du dépôt distant vous demandera votre clé SSH **publique** afin de mettre à jour les autorisations sur `git`.

Voir : http://git-scm.com/book/fr/Les-bases-de-Git-Travailler-avec-des-d%C3%A9p%C3%B4ts-distants

git remote

Il est intéressant de mémoriser sous la forme d'alias les adresses internets des dépôts distants.

```
$ git remote add <alias> <url> #syntaxe de git remote

#exemple
$ git remote add plum git@37.187.66.61:/plummvc.git #crée l'alias plum

#liste toutes les alias
$git remote -v

#informations sur un dépôt distant
$ git remote show [nom-distant] #la commande
$ git remote show origin #informaton sur le dépôt par défaut
$ git remote show plum #informaton sur le dépôt plum
```

- Par défaut l'alias d'un dépôt se nomme `origin`

git clone

L'autre façon de démarrer un dépôt gite consiste à cloner un projet existants.

positionnez vous sur le répertoire qui contiendra le dépôts cloné, par exemple `c:/git/`

```
$ git clone <nom distant> #nom distant :  une url

$ git clone git@37.187.66.61:/plummvc.git #clone le dépôt distant
#clone le dépôt distant dans le répertoire c:/git/plummvc/
```

- Le répertoire `c:/git/plumvc/` contient une copie du dépôt distant.

git push

Pour mettre à jour un dépôt distant on *pousse* le dépôt local avec `git push`

```
$ git push <nom distant> <nom de branche> #commande git push

#exemple
$git push git@37.187.66.61:/plummvc.git master #url
git@37.187.66.61:/plummvc.git
$ git push plum master #alias plum

##toutes les branches
$ git push plum --all;
```

- **master** est le nom de la branche principale d'un dépôt

https://www.atlassian.com/fr/git/tutorial/remote-repositories#!push

git pull

`git pull` met à jour votre dépôt local à partir du dépôt distant.

Récupère par `fetch` la copie de la branche active dans le dépôt distant spécifié, et la fusionne immédiatement par `merge` dans la copie locale. *Identique à git fetch <distant>, suivi de git merge origin/<branche-actuelle>.*

```
$ git pull <serveur distant> #git pull

#exemple
$ git pull plum #met à jour la branche active depuis le dépôt distant plum
```

- La branche active par défaut est `master`. La branche active change au moment d'un `git checkout`.

Voir :

- http://git-scm.com/book/fr/Les-branches-avec-Git-Les-branches-distantes
- http://doc.ubuntu-fr.org/git#recuperation_des_changements

git fetch

Tirer un dépôt distant sans le fusionner avec le dépôt existant sur votre machine.

Voir :
http://git-scm.com/book/fr/Les-bases-de-Git-Travailler-avec-des-d%C3%A9p%C3%B4ts-distants#R%C3%A9cup%C3%A9rer-et-tirer-depuis-des-d%C3%A9p%C3%B4ts-distants

Branches

git branch

Il est possible de gérer un dépôt sous forme de branche.

Fix Me! la branche principale s'appelle `master`.

```
#forme générale
$git branch <mabranche> [<tag ou commit ou nom de branche>]

#liste des branches. * devant la branche active
$ git branch -v

#Supprimer une branche
$ git branch toto -D #supprime la branche toto

#Travailler avec une branche
$git checkout <mabranche> #permet de travailler avec <mabranche> [devient
HEAD]
```

- **On peut créer une branche** à partir d'un tag ou d'un commit ou d'une branche

Exemples :

```
$ git branch b1 t1 #création de la branche b1 à partir du tag t1
$ git checkout b1 #positionnement sur la branche b1

#schéma
!
!
!(tag t1)---branch t1
!
!(commit)
!
(HEAD)
```

git checkout

`git checkout` permet de désigner une branche comme **active**.

exemple :

```
$ git checkout ae6788e #se positionne sur le commit commençant par ae6788e

$ git checkout master #on revient à l'état du dépôt le plus récent

$ git checkout tag1 #branchement vers le tag tag1
```

```
$ git branch #liste des branches: * devant la branche active
```

Fix Me! `git checkout` modifie les fichiers du dépôts. Les fichiers sont dans l'état du `commit` ou de la branche ou du tag indiqué par `git checkout`.

Fix Me! Si on veut retrouver l'état de nos fichiers plusieurs `commit` en arrière, il existe la version *j'écrase tout et je recommence* avec `git reset`….

Mais on peut aussi grâce à `git checkout` revenir en arrière en gardant ce qui a été fait.

git add : git commit

La commande `git add` permet d'indexer les fichiers de votre dépôt dans git. L'indexation sera définitive quand vous aurez réalisé la commande `git commit`.

```
#Pour connaitre l'état de votre dépôt
$ git status #pour savoir où l'on en ait des modifications

#indexer tous les fichiers avec les sous-répertoires
$ git add *
$ git add .
$ git add --all

$ git add index.php -v #indexer le fichier index.php, option verbose

#Valider vos modifications
$ git commit -m 'votre message' #validation indexage

#Liste des commits
$ git log #liste des commits

$ git log --pretty=oneline #affiche chaque commit sur une ligne
ca82a6dff817ec66f44342007202690a93763949 changed the version number
085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7 removed unnecessary test code
a11bef06a3f659402fe7563abf99ad00de2209e6 first commit
```

Fix Me! `git commit` n'intégre que les modifications indexées par `git add`.

Voir :
http://git-scm.com/book/fr/Les-bases-de-Git-Enregistrer-des-modifications-dans-le-d%C3%A9p%C3%B4t#Valider-vos-modifications

git tag

```
$ git tag montag -m 'mon message' #crée un tag sur le dernier commit

$ git tag montag <commit> #crée un tag sur le ''commit'' indiqué
```

```
$ git tag -n #affiche la liste des tags et le message attenant
```

git revert

```
$ git revert b12caf55 #défait les modification du commit b12caf55...
```

git reset

```
$ git reset --hard <commit1> #remet la branche courante à l'état du commit
donné en paramètre.
$ git reset --soft <commit2> #remet la branche courante à l'état du commit
mais laisse les différences
```

- l'option **-hard-** efface TOUS les `commit` réalisé après `<commit1>`

Voir : http://gitref.org/index.html

Au secours !

Récupération de données

Mauvaises manipulations : effacement, ajout intempestif …

Allez ici : http://git-scm.com/book/fr/Les-tripes-de-Git-Maintenance-et-r%C3%A9cup%C3%A9ration-de-donn%C3%A9es ssh

SSH

Cette partie est consacrée à l'usage du protocole SSH pour la mise à jour de **dépôts distants**.

Le protocole SSH est le protocole de communication privilégié par **git** pour le gestion de dépôt distant.

Pour la sécurité des échanges il est nécessaire que chaque intervenant sur un dépôt dispose **d'une clé personnelle**.

- une clé publique qui sera sur le serveur du dépôt
- une clé privée qui ne devra être communiqué à personne

ssh-keygen

La commande `ssh-keygen` crée le répertoire `.ssh`.

Le répertoire `.ssh` contient

- la clé privée : **id_rsa** est le fichier contenant la **clé privée**
- la clé publique : **id_rsa.pub** est le fichier contenant la **clé publique**

Pour approfondir :

- https://help.github.com/articles/generating-ssh-keys
- https://gist.github.com/jexchan/2351996
- https://help.github.com/articles/working-with-ssh-key-passphrases

Générer une clé SSH

Avec l'installation de `git pour Windows` vous disposez de la commandes `ssh-keygen`.

Avec Windows les données SSH sont mémorisées dans

```
c:/USERS/nomUtilisateur/.ssh/
```

Démarche

Cette façon de faire est celle préconisée par github

```
$ ssh-keygen -C "votreAdresseMail"

Enter file in which to save the key
(/c/Users/nomUtilisateur/.ssh/id_rsa):<enter>

Enter passphrase (empty for no passphrase):<ma phrase secrète>
Enter same passphrase again:<ma phrase secrète>

Your identification has been saved in /c/Users/nomUtilisateur/.ssh/id_rsa.
Your public key has been saved in /c/Users/nomUtilisateur/.ssh/id_rsa.pub.

The key fingerprint is:
d0:71:98:a3:16:55:88:8b:d5:bd:e2:8a:0a:41:68:d2 nomUtilisateur@WIN-
JQFKJHT191F
```

- **ssh-keygen** : Permet de générer le calcul de la clé privée et de la clé publique depuis `Git Bash`
- **Enter file...** : Choisir le répertoire contenant les `clés ssh`
- **PassPhrase** : Choisir une phrase pour le mot de passe

Votre clé privée doit être tenue **secrète**. Il est recommandé de la protéger par une phrase servant de mot de passe.

Git et SSH

Par défaut `Git` recherche votre clé privée dans le répertoire `c:/users/nomUtilusateur/.ssh/`.

- A chaque accès au dépôt distant la clé privé sera fourni par **Git**.

PassPhrase

L'utilisation de la `passphrase` est demandée à chaque fois que vous voulez accéder au dépôt central.

.profile

Pour éviter de retaper la `passphrase` vous devez mettre le fichier `.profile` dans le répertoire `c:/users/nomUtilisateur/`.

- Copier et coller les lignes suivantes dans le fichier `c:/USERS/nomUtilisateur/.profile`.

Utiliser `NotePad++` pour pouvoir mémoriser correctement votre fichier.

```
# Note: ~/.ssh/environment should not be used, as it
#       already has a different purpose in SSH.

env=~/.ssh/agent.env

# Note: Don't bother checking SSH_AGENT_PID. It's not used
#       by SSH itself, and it might even be incorrect
#       (for example, when using agent-forwarding over SSH).

agent_is_running() {
    if [ "$SSH_AUTH_SOCK" ]; then
        # ssh-add returns:
        #   0 = agent running, has keys
        #   1 = agent running, no keys
        #   2 = agent not running
        # if your keys are not stored in ~/.ssh/id_rsa.pub or
~/.ssh/id_dsa.pub, you'll need
        # to paste the proper path after ssh-add
        ssh-add -l >/dev/null 2>&1 || [ $? -eq 1 ]
    else
        false
    fi
}

agent_has_keys() {
    # if your keys are not stored in ~/.ssh/id_rsa.pub or ~/.ssh/id_dsa.pub,
you'll need
    # to paste the proper path after ssh-add
    ssh-add -l >/dev/null 2>&1
}
```

```
agent_load_env() {
    . "$env" >/dev/null
}

agent_start() {
    (umask 077; ssh-agent >"$env")
    . "$env" >/dev/null
}

if ! agent_is_running; then
    agent_load_env
fi

# if your keys are not stored in ~/.ssh/id_rsa.pub or ~/.ssh/id_dsa.pub,
you'll need
# to paste the proper path after ssh-add
if ! agent_is_running; then
    agent_start
    ssh-add
elif ! agent_has_keys; then
    ssh-add
fi

unset env
```

Au démarrage de `Git Bash` le fichier `.profile` est exécuté. `ssh-agent` vous demande votre phrase et la conserve la durée de votre connexion.

ssh-add

```
 $ ssh-add
```

permet de demander la mémorisation de la `passphrase` pour l'accès à votre clé privée SSH.

Cela évite de la resaisir à chaque `push` ou `pull`.

Voir : https://help.github.com/articles/working-with-ssh-key-passphrases

Liens

- http://git-scm.com/book/fr/D%C3%A9marrage-rapide
- http://git-scm.com/
- https://help.github.com/
- https://help.github.com/articles/duplicating-a-repository
- https://gist.github.com/jexchan/2351996

bad file number

CETTE PARTIE N'EST PLUS D'ACTUALITE MAIS SUBSISTE UNIQUEMENT POUR INFORMATION

L'erreur suivante est souvent du à un proxy :

```
ssh: connect to host github.com port 22: Bad file number
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
```

Au lycée il n'est pas possible d'utiliser `ssh` en natif pour les mises à jour vers `github`. La solution consiste à ajouter un fichier `config`.

```
#~/.ssh/config

Host github.com
  Hostname ssh.github.com
  Port 443
```

Voir :

- http://www.git-attitude.fr/2010/09/13/comprendre-et-maitriser-les-cles-ssh/
- https://help.github.com/articles/using-ssh-over-the-https-port
- https://help.github.com/articles/error-bad-file-number

From:
http://ppe.boonum.fr/ - **AP.SIO**

Permanent link:
http://ppe.boonum.fr/doku.php?id=outil:git:documentation&rev=1511431597

Last update: **2017/11/23 11:06**