> Notes :

-

1 > Mise en route

> Configuration

- git config --global user.email marc-nathan@hotmail.fr

- git config --global user.name 'Marc NATHAN'

- git config --global -e

Note : en omettant l'option --global il est possible d'utiliser une identité différente pour un dép particulier

> Création d'un dépôt

- git init // pour initialiser un projet en local dans le répertoire de W

- git init --bare deopot.git // pour initialiser un projet sur le server de partage

> Téléchargement d'un dépôt existant

- git clone [ssh|git|http|ftp]: //login@git.exemple.com/chemin/depot.git [depot ou répertoire de W]

2 > Modification (commandes de bases)

> git add <fichier> // Ajout du fichier à l'index

> git stage <fichier> // Ajout du fichier à l'index

// Pratique pour changer de branche

// Au retour faire git <fichier> apply

> git add -p <fichier> // Ajout des chamgements d'un fichier à l'index

> git stage -p <fichier> // Ajout des chamgements d'un fichier à l'index

> gir rm <fichiers> // Supprime un fichier de l'index et du répertoire de W

> git rm --cached <fichier> // Supprime un fichier de l'index

> git rm -r <répertoire> // Supprime un répertoire de l'index et du répertoire de W

> git rm -r --cached <répertoire>// Supprime un répertoire de l'index

> git mv <ancien> <nouveau>// Renommage de fichier

> git commit // enregistre le contenu de l'index dans un nouveau commit

> git commit -a // Enregistre le le contenu du répertoire de W dans un nouveau commit.

// appelle git rm sur tous les fichiers supprimés at add sur les modifiés

> git commit <fichier> // Créer un nouveau commit contenant uniquement les changements conernant un fichiers

> git commit --amend // Enregistre le contenu de l'index dans le dernier commit (head)

// A ne pas utiliser si on a déjà pushé

> git citool // Ouvre un gui pour gérer les commits

> git status // Affiche l'état des fichiers du répertoire de W et de l'index

3 > Exploration (changements, historique)

> git diff // Affiche la différence entre le contenu de l'index et le répertoire de W

> git diff --cached // Affiche la différence entre le dernier commit (HEAD) et l'index (add)

> git diff HEAD // Affiche la différence entre le dernier commit (HEAD) et le répertoire de W

> git diff A B // Affiche la différence entre le contenu pointé par A et celui de B

> git log [A] // Affiche l'historique des changements (listes de commits) accessibles depuis le commit (HEAD) pointé par A

// Si A est omis, c'est HEAD qui est employé

> git log [A] --<fichier> // Idem en restreignant l'affichage aux commits qui modifient le fichiers indiqué

// Très pratique pour voir les derniers changements ayant affecté un fichiers

> git log A..B // Affiche l'historique entre deux commits (HEAD)

> git log --graph --online [A] // Affiche une représentation textuelle de l'historique accessible à partir de A(HEAD si omis)

> git blame [A] [--] <fichier> // Affiche le contenu du fichier tel que présent dans l'arborescence de A(HEAD si omis) en indiquant

// pour chaque ligne le dernier commit à avoir modifié (ou introduit) celle-ci

> git log --stat // Affiche un diffstat correspondant aux changements du commit : la liste des fichiers modifiés

// avec les statistiques sur le nombre de lignes ajoutée et supprimées

> Schémas du mécanisme

répertoire de W ndex Historique (HEAD)

propagation giot add <fichers>

du contenu ------------------------------->

git commit

------------------------>

git commit (-a | <fichiers>)

-------------------------------------------------------->

git reset -- <fichiers>

<------------------------

git checkout -- <fichiers>

<-------------------------------

git checkout HEAD -- <fichiers>

<--------------------------------------------------------

Analyse des git diff --cached

différences <------------------------

git diff

<-------------------------------

git diff HEAD

<--------------------------------------------------------

4 > Annulation, restauration et nettoyage

> git reset // Efface les changements en attente de commit (réinitialisation de l'index (add) avec

// le contenu de HEAD. Fichiers du répertoire de W inchangé

> git reset --hard // Effece tous les changements présents dans l'index et le répertoire de W avec le contenu de HEAD

// Attention cette commande est destructrice

> git reset [--] <fichier> // Réciproque de git add, efface uniquement les changements en attente de commit qui affectait le <fichier>

//partir de Head, l'état dans l'index est retauré

> git checkout <fichier> // Restaure le <fichier> du répertoire de W à son état dans l'index

> git checkout HEAD [--] <fichier>// restaure le <fichier> du répertoire de travail et de l'index à son état dans HEAD

> Notre : pour les trois dernières commandes, le -p permet de choisir intéractivement le sous-ensemble des changements à effacer

> git reset A [--] <fichier> // Restaure l'état du <fichier> dans l'index, à son état dans le commit

> git checkout A [--] <fichier>// Restaure l'état du <fichier> dans le répertoire de W et de l'index à partit du commit A

> Revenir sur des changements compmités

Commande Commit HEAD Index Répertoire de W

git reset --soft A A A inchangé inchangé

git reset --meduim A A A A inchangé

git reset A

git reset --hard A A A A

> git clean // Sert à supprimer du répertoire de travail les fichiers non suivis (et non ignoré) par git

> git clean -f // Il faut systématiquement passer l'option -f pour réaliser cette opération desctructrice

// Sauf si l'option clean.requireForce est à False

> git clean -n // Affiche ce qui aurait été supprimé

5 > Gestion des branches

> git branch // Liste toutes les branches locales

> git branch -r // Liste toutes les branches distantes

> git branch -a // Liste toutes les branches locales et distantes

> git branch <branche> [A] // Créer une nouvelles <branche> pointant sur A, ou HEAD si A est omis

// Attention, cette commande ne modifie pas la branche courante

> git checkout A // Fait de A la branche courante et met à jour le répertoire de W.

// Cette opération peut échouer si des changements non commités ne peuvent être conservé au sein de A

> git checkout -b <branche> [A] // Créer une nouvelle branche <branche> pointant sur A, ou sur HEAD si A est omis, puis fait de <branche>

// la branche courante et met à jour le répertoire de W

> git merge A // Fusionne A dans la branche courante. Habituellement cette commande créer un nouveau commit représentant

// le résultat de cette fusion, celui-ci disposant alors des deux commits parents

// Si A contient déjà tous les changements de la branche courante, la fusion se fait par <fast foward> :

// la branche courante est simplement mise à jour pour pointer vers A, aucun nouveau commpit n'est créer

> git merge -s ours A // Fusionne A dnas la branche courante, mais en ignorant tous changements qu'elle introduit, juste pour

// créer le point de parenté

// Utile lorsqu'un correctif est introduit dans une très vielle branche, et qu'il est déjà présent voir inutile

// dans la branche de développement courante, pour fusionner cette ancienne branche sans avoir à générer des conflits pléthoriques

> git mergetool // Outil facillitant la résolution des conflits qui peuvent survenir lors de la fusion de branches

> git branch -d <branche> // Supprime la branche <branche>. Cette commande échoue si <branche> n'est pas accessible dans l'historique de la branche courante.

// On peut toutefois forcer cette suppression en employant -D au lieu de -d

> git branch -m [ancien] nouveau// Renomme la branche <ancien> (ou la branche courante si ancien est omis) en <nouveau>

// Cette commande échoue si <nouveau> existe déjà. On peut toutefois écraser <nouveau> en utilisant -M au lmieu de -m.

6 > Résolution de conflits

> git remote // Liste les dépôts distant configurés

> git remote add <dépôt> url // Ajout un nouveau <dépôt> distant accessible à l'URL induquée

> git remote update // Met à jour la copie locale de tous les dépots distants en exécutant dit fetch sur chacun des dépôts configurés.

// Avec l'option --prune, les branches distantes ayant disparus du dépôts distant sont également supprimées.

> git fetch <dépôt> // Met à jour la copie locale du dépôt distant

> git push [<dépôt>] // Envoie les changements des branches locales vers les éventuelles branches distantes de même nom du <dépôt> distant

// Si omis c'est le dépôt distant associé à la branche courante qui est employé, sinon c'est origin qui est employée

> git push <dépôt> L:D // Créer ou met à jour la branche distante S avec les changements contenusdans la branche locale L

> git push L // Si " :D " est omis, alors la branche distante a le me nom que la branche locale L

> git push :D // Si L est omis, la branche distante D est supprimée

> git pull // Met à jour la branche distante que la branche courante suit, puis intègreles nouveaux changements dans la branche courante.

// Avec l'option --rebase, la mise à jour de la branche locale est effectuée avec git rebase branche-distante au lieu de git merge branche-distante

7 > Partage et publication

> git format-patch A // Extrait tous les commits de la branche courente qui ne nont pas dans la branche A sous forme de patch stockés dans le répertoire

// courant (ou dans le répertoire précisé avec l'option -o ou --output-directory)

> git send-email <fichier> // Envoie tous les <fichiers> patch

> git send-email <répertoire> // (ou tous les fichiers patch contenu dans le <répertoire> par courrier électronique

> git send-email <branch> // Utilise directement git-format-patch <branch> pour générer les patchs à envoyer

// De nombreuses options existent pour définir les différents champs du courrier expédié.

// Dont l'option --to qui indique le destinataire principale

> gir push // Envoie d'une branche sur un dépôt

> git tag <nom> // Créer un tag et l'associe au commit courant. Un tag permet de retrouver facilement l'état du projet à un moment donnée cléf

// lorsqu'une nouvelle version est publiée. L'option -s permet de signer un tag avec GniPG.

// L'option -d supprime le tag nommé.

> git archive [--prefix=valeur]

-o <fichier>.tar.gz A // Créer une archive fichier.tar.gz correspondant à une copie du contenu du dépôt tel qu'il est dans le commit pointé par A.

// Il est possible de créer de .zip

// L'option --prefix définit un préfixe à ajouter à tous les noms de fichiers. En choisissant " projet/ " comme préfixe,

// on place tous les fichiers dans un répertoire " projet " au sein de l'archive générée

> git apply <patch> // Applique le patch au répertoire de W. Si --inedex est passé, applique également le patch à l'index.

// Si --cached est passé, alors le patch n'est appliqué qu'à l'index est pas au répertoire de W.

// Cette commande ne créer pas de commite.

8 > Modifications (commandes avancées)

> git am <boite au lettre> // Applique une série de patch reçus par mail et créer un commit pour chaque patch.

> git am <fichiers patch>

> git cherry-pick A... // Pour chaque commit indiqué (A...) créer un nouveau commit dans la branche courente et duplique le contenudu commit indiqué.

// Alors qu'une fusion intègre tous les changements d'une branche, cette commande permet de choisir exactement quelschangements

// intégrer dans la branche courante.

> git revert A... // Pour chaque commit indiqué (A...), créer un nouveau commit dans la branche courante qui annule les changements apportés par la commit indiqué.

// Réciproque de git cherry-pick.

> git rebase A // Réécrit la branche courante de telle manière que tous ses commits soient des descendant de A. Pour cela, tous les commits de la branche courante

// (depuis le premier descendant du premier ancêtre commun à A jusqu'à HEAD) sont recréees (comme avec git cherry-pick).

// LE plus ancien commit devient le descendant direct de A.

> git rebase --onto B A // Réécrit la branche courante (dérivée de A) de telle manière que tous ses commits coient des descendants de B. Pour cela, tous les commits de la branche

// courante (depuis le premiers descendant du premier ancêtre commun à A jusqu'à HEAD) sont recréés (comme avec git cherryèpick).

// Le plus ancien commit devient le descendant direct de B.

> git rebase -i A // L'option -i permet de modifier de manière interactive l'historique des changements en réordonnant les commits, en fusionnant certains commits dans d'autres

// (squash, fixup), en modifiant ou découpant des commits (edit), voir en en supprimant certains, git commit --fixup=B (ou --squash=B) créer un commit marqué

// comme étant à fusionner dans B au cours du prochain git rebase --autosquash -i qui inhcluera ce commit.

// La différence entre squash et fixup est que le premier ouvre un éditeur pour modifier le message de commit résultant de la fusion,

// alors que le second conserve le message de commit ciblé.

>

9 > L'arme ultime pour isoler le commit fautif : git bisect

>

>

>

>

>

>

11 > Fichiers de configuration

>

>

>

>

>

>

12 > Principales options de configuration de Git

>

>

>

>

>

>

13 > Alias fréquemment employés

>

>

>

>

>

>