LEXIQUE

Dépôt / Repo(sitory)
Base de données, généralement stockée dans le dossier .git à la racine du working directory, qui contient l'ensemble des objets et références utilisés par Git pour la gestion de sources de votre projet.

Remotes
Références vers les dépôts distants, qui servent de points d'échange avec les autres collaborateurs.

Snapshot
Photographie instantanée d'un contenu. Git utilise des snapshots des fichiers qu'il versionne, et non des difs, ce qui permet d'accélérer énormément de nombreux traitements.

SHA1 (« Chawan »)
Identifiant unique, d'un objet Git (commits, tags, etc.), calculé à partir d'un algorithme de hashing (calcul d'une valeur succincte unique à partir d'une masse quelconque de données). Le SHA1 est constitué de 40 caractères hexadécimaux.

Commit
Objet Git constitué d'un snapshot de votre code source au moment ou vous avez souhaitez enregistrer sa version. Le commit comporte également un SHA1 et un ensemble de Métadonnés (auteur, date, …).

WD / Working Directory
Dossier contenant votre code source géré en version par Git À sa racine, on trouve normalement le dossier .git qui contient le dépôt associé. Un dépôt « côté serveur » n'a pas besoin de working directory associé : on parle de dépôt nu (bare repo).

Stage / Index
Zone intermédiaire entre le working directory et le dépôt local. L'opération de commit ne prend que ce qui est dans le stage, et l'opération git add place des contenus dans le stage. Cette zone permet de « sculpter » le prochain commit, pas à pas.

Stash
Parfois traduit par « remise », c'est une zone annexe, locale au dépôt, en marge de l'historique et des branches, une sorte de presse-papier multiple orienté Git, qui permet de mettre en suspens un travail en cours (fichiers pas encore versionnés, modifications locales, stage) pour revenir à un état net, et de reprendre ces travaux ultérieurement.

Historique / log
Déroulé public de la séquence des commits qui constituent le dépôt au fil du temps. Affiché par défaut en ordre chronologique inverse (le plus récent d'abord). En raison des branches et fusions, constitue en fait un graphe plus qu'une simple ligne droite, avec plusieurs axes de travail parallèles pendant un temps.

Tag
Étiquette apposée sur un commit, pour l'identifier de manière plus humainement compréhensible qu'un SHA1. N'est pas censé bouger après avoir été défini, contrairement aux pointes de branches, qui évoluent dans le temps.

HEAD
Pointeur sur un commit de référence à l'instant présent, vis-à-vis duquel le stage et le working directory sont comparés.

ORIG_HEAD
Pointeur sur le commit pointé précédemment par HEAD.

Reflog
Historique strictement local et privé des positions successives, dans le dépôt local courant, du HEAD.

Tête détachée (detached head)
État dans lequel le HEAD ne référence pas une branche par son nom : il référence directement un SHA1. Dans cet état, un commit reste possible mais se retrouve isolé, faute de branche active. Survient notamment dès qu'une opération (checkout, fusion, rebase, submodule update...) se base sur un SHA1 plutôt qu'un nom de branche.

Reset
Déplacement explicite du HEAD et de la branche courante sur un commit donné.

Fusion / Merge
Action de rapatrier le travail issu d'une branche dans une autre.

Conflit
Lors de la fusion de deux commits ou branches (fusion, cherry-pick, rebase, stash pop...), situation dans laquelle Git n'arrive pas tout seul à arbitrer la fusion des modifications de part et d'autre sur un même fichier.

Rebase
Action de réécrire l'historique d'un commit, généralement une branche entière, soit en la déplaçant (en changeant son point de base : re-base), soit en réécrivant simplement son historique propre (pour le nettoyer), soit les deux.

INSTALLATION

Outil Git + interphases graphique (git et gitk).

Linux: sudo apt-install git-all
Mac/Windows: https://git-scm.com/download

COMMANDES

CONFIGURATION

Configurer git pour le dépôt courant ou pour tous avec l'option --global

$ git config user.name "[nom]"
Définit le nom de l'utilisateur

$ git config user.email [email]
Définit l'email de l'utilisateur

$ git config core.editor <editor>
Définit l'éditeur de texte par défaut pour Git

GESTION DES DÉPÔTS

Créer un dépôt local ou en utiliser un distant via son url.

$ git init --bare [nom_dépôt]
Créer un dépôt local vide dans le dossier courant ou en créer un avec le nom spécifié. --bare permet de créer un dépôt sans workspace.

$ git clone [url] [nom_dépôt]
Duplique en local le dépôt git pointé par l'url. Le dépôt local comportera alors un remote du nom de origin.

$ git remote -v
Liste les remotes du dépôt courant.

$ git remote add [remote_name] [remote_url]
Ajoute un remote au dépôt courant.

$ git remote rm [remote_name]
Supprime le remote du dépôt courant.

$ git remote rename [old_name] [new_name]
Renomme le remote ciblé.

SYNCHRONISER LES CHANGEMENTS

Synchroniser le dépôt local et distant. En l'absence de remote dans la commande, c'est origin qui est utilisé.

$ git fetch [remote]
Met à jour le dépôt locales avec les informations du serveur distant mais ne les merge pas avec la branche courante et le workspace.

$ git pull [remote] [branch]
Met à jour le dépôt locales avec les informations du serveur distant et les merge avec la branche courante et le workspace.

$ git push [remote] [branch]
Synchronise le serveur distant et le remote. Permet d'envoyer les modifications locales vers le remote.

GÉRER LES MODIFICATIONS

Voir les changements, les sélectionner et les enregistrer.

$ git status
Donne l'état courant du workspace et dépôt local.

$ git checkout [fichier]
Supprime les modifications courantes du fichier. Attention, impossible de les récupérer après

$ git add --patch [fichiers]
Ajoute les modifications d'un fichier a la zone d'index. Mode interactif avec --patch

$ git reset HEAD [fichiers]
Enlève, de la zone d'index, les modifications mais les conserves dans le workspace.

$ git diff --staged [fichier]
Montre les modifications du fichier non indexé, pour un fichier indexé, ajouter l'option --staged.

$ git commit --amend -m "[message]"
Enregistre les modifications présentes dans la zone d'index dans l'historique du dépôt. --amend permet de modifier le commit précédent.

$ git rm --cached [fichier]
Supprime du suivi de git le fichier. Ce fichier est également supprimé du WD sauf avec --cached

$ git mv [src] [dest]
Déplace/Renomme un fichier dans le WD tout en conservant le suivi de son historique.

GESTION DE L'HISTORIQUE

Visualiser et naviguer dans l'historique du dépôt Git.

$ **git log [-n]** --oneline --decorate --graph --all
Affiche l'historique des commits depuis la position du HEAD.. -n pour limiter aux n derniers commits.
--oneline: pour limiter à une ligne par commit.
--graph: pour afficher sous forme d'arbre.
--decorate: pour afficher avec de la couleur.
--all: pour afficher toute les branches.

$ git diff [tag|sha1] [tag|sha1]
Montre les différences de contenu entre deux commits.

$ git show [tag|sha1]
Montre les modifications enregistrés lors du commit spécifié.

$ git checkout [tag|sha1|branche]
Se déplacer dans l'historique sur le commit donné

$ git tag -n
Liste les tags présents dans le dépôt local.

$ git tag -d **[nom_tag]**
Créer un tag sur le commit courant. L'option -d permet de supprimer le tag.

RÉÉCRIRE L'HISTORIQUE

Corriger des erreurs, regrouper ou modifier des commits.
$ git commit --amend
Modifier le commit précédent (contenue et message)

$ git revert [tag|sha1|branche]
Créer un nouveau commit qui est l'opposé du commit spécifié afin d'annuler ses effets.

$ git reset [--soft|--mixed|--hard] **[tag|sha1|branch]**
Déplace HEAD et la tête de la branche courante. l'option par défaut est **--mixed**
-- soft: conserve le workspace et la zone d'index
-- mixed: conserve le workspace uniquement
-- hard: ne conserve pas le workspace et l'index
$ git rebase –i [tag|sha1|branch]
Réordonne, fusionne, supprime, modifie les n dernier commits local. A ne pas faire sur des commit poussé sur le remote.

TRAVAILLER EN BRANCHE

Créer et gérer les branches en local et via les remotes

$ git branch -u [remote/branch] [branch]
Fait suivre une branche distante par une branche locale.

$ git branch -r -a
Liste toutes les branches locales dans le dépôt courant. **-r** pour les branche distantes. **-a** pour tout

$ git branch [nom_branche]
Crée une nouvelle branche.
git branch -d [nom_branche]
Supprime la branche local.

$ git checkout -b **[nom_branche]**
Change de branche et ce placer sur le dernier commit de celle-ci. **-b** pour en plus créer la branche.

$ git merge [autre_branche]
Combine dans la branche courante l'historique de la branche spécifiée via un commit de merge.

$ git rebase [autre_branche]
Déplace les commits de la branche courante sur la branche spécifiée.

$ git cherry-pick [-x] [commit]
Applique un commit à l'espace de travail. -x permet d'ajouter le message « cherry-picked from commit [sha1_commit] ».

METTRE DE CÔTÉ DES MODIFICATIONS

Mètre de côté des modifications temporairement.

$ git stash save "[message]"
Enregistre toutes les modifications courante dans une pile temporairement.

$ git stash list
Liste toutes modifications mises de côté.

$ git stash pop id
Appliquer les modifications à l'index id de la pile dans l'espace de travail. sinon id = 0

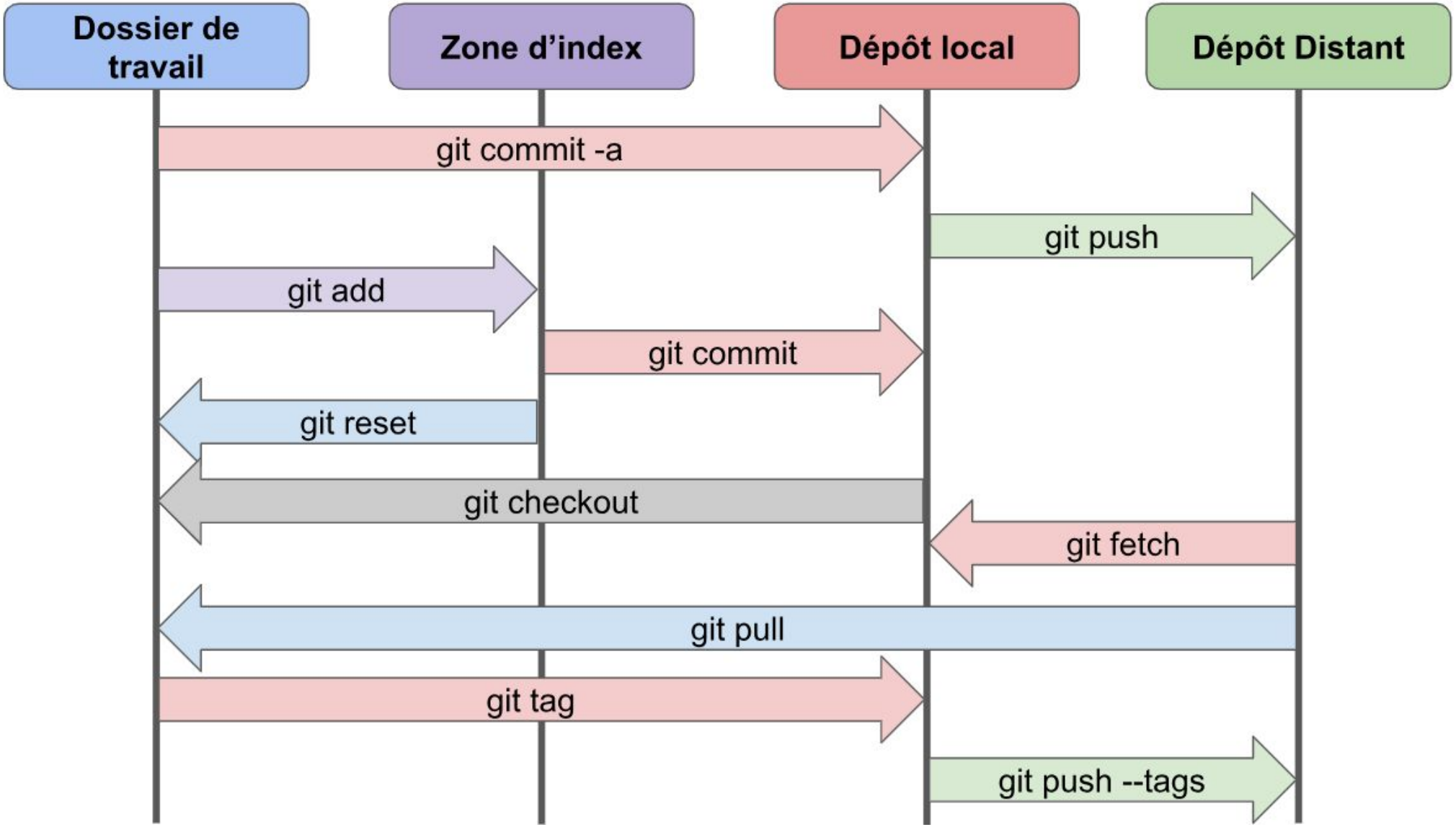
$ git stash drop id
Supprime les modifications à l'index id de la pile.
sinon id = 0.

DEBUGER SON CODE

Outils d'aide au debug.

$ git blame -L [line_start],[line_stop] [file_name]
Affiche par qui et quand chaque lignes du fichier a été modifié.

$ git bisect [commit_bad] [commit_good]
Recherche par dichotomie l'origine d'un bug entre deux commits.



Formateur: Arnaud MERCIER
Email: arnaud.mercier.formation@gmail.com
Site: www.codeur-pro.fr

Codeur Pro