Git Aide-Mémoire

TICHANE Jean-Michel

01 Configuration de Git

```
$ git config --global user.name “Your Name”
```

Définissez le nom qui sera attaché à vos commits et tags.

```
$ git config --global user.email “you@example.com”
```

Définissez l'adresse e-mail qui sera jointe à vos commits et tags.

```
$ git config --global color.ui auto
```

Activez une certaine colorisation de la sortie Git.

02 Commencer un projet

```
$ git init [project name]
```

Créez un nouveau référentiel local. Si **[project name]** est fourni, Git créera un nouveau nom de répertoire **[project name]** et initialisera un référentiel à l'intérieur. Si **[project name]** n'est pas fourni, un nouveau référentiel est initialisé dans le répertoire courant.

```
$ git clone [project url]
```

Télécharge un projet avec l'intégralité de l'historique à partir du référentiel distant.

03 Travail au quotidien

```
$ git status
```

Affiche l'état de votre répertoire de travail. Les options incluent les fichiers nouveaux, préparés et modifiés. Il récupérera le nom de la branche, l'identifiant de validation actuel et les modifications en attente de validation.

```
$ git add [file]
```

Ajoutez un fichier à la **staging area (zone de préparation)** . Utilisez à la place du chemin d'accès complet au fichier pour ajouter tous les fichiers modifiés du **répertoire en cours** dans **l'arborescence des répertoires.**

```
$ git diff [file]
```

Afficher les changements entre le **working directory** et la **staging area**.

```
$ git diff --staged [file]
```

Affiche toutes les modifications entre la **staging area** et le **repository**.

```
$ git checkout -- [file]
```

Ignorer les modifications dans le **working directory**. Cette opération est **unrecoverable**.

```
$ git reset [file]
```

Rétablissez votre **repository** à un état de fonctionnement antérieur connu.

```
$ git commit
```

Créez un nouveau **commit** à partir des modifications ajoutées à la **staging area**. Le **commit** doit avoir un message !

```
$ git rm [file]
```

Supprimer le fichier du **working directory** et de la **staging area**.

```
$ git stash
```

Mettez les modifications actuelles de votre **working directory** dans la **stash (cachette)** pour une utilisation ultérieure.

```
$ git stash pop
```

Appliquez le contenu de **stash** dans le **working directory**, et effacez la **stash**.

```
$ git stash drop
```

Supprimez une **stash** spécifique de toutes vos **stashes** précédentes.

04 Modèle de branchement Git

```
$ git branch [-a]
```

Répertorier toutes les branches locales dans le référentiel. Avec **-a**: affiche toutes les branches (avec remote).

```
$ git branch [branch_name]
```

Créez une nouvelle branche, référençant le **HEAD** actuel.

```
$ git checkout [-b][branch_name]
```

Bascule le **working directory** vers la branche spécifiée. Avec **-b**: Git créera la branche spécifiée si elle n'existe pas.

```
$ git merge [from name]
```

Joindre la branche **[from name]** spécifiée à votre branche actuelle (celle sur laquelle vous vous trouvez actuellement).

```
$ git branch -d [name]
```

Supprimer la branche sélectionnée, si elle est déjà fusionnée avec une autre.
-D au lieu de **-d** fforce la suppression.

05 Passez en revue votre travail

```
$ git log [-n count]
```

Liste l'historique des commits de la branche actuelle. **-n count** limite la liste aux **n** derniers commits.

```
$ git log --oneline --graph --decorate
```

Une vue d'ensemble avec des étiquettes de référence et un graphique d'historique. Un commit par ligne.

```
$ git log ref..
```

Répertorier les commits présents sur la branche actuelle et non fusionnés dans **ref**. Une **ref** peut être un nom de branche ou un nom de balise.

```
$ git log ..ref
```

Répertorier les commits présents sur **ref** et non fusionnés dans la branche actuelle.

```
$ git reflog
```

Répertorier les opérations (par exemple, les extractions ou les validations) effectuées sur le référentiel local.

06 Balisage des commits connus

```
$ git tag
```

Lister toutes les balises.

```
$ git tag [name] [commit sha]
```

Créez une référence de balise nommée **name** pour la validation actuelle. Ajoutez **commit sha** pour baliser un commit spécifique au lieu du commit actuel.

```
$ git tag -a [name] [commit sha]
```

Créez un objet tag nommé **name** pour le commit actuel.

```
$ git tag -d [name]
```

Supprimer une balise du référentiel local.

07 Annuler des changements

```
$ git reset [--hard] [target reference]
```

Bascule la branche actuelle vers la **target reference**, en laissant une différence en tant que modification non validée. Lorsque **--hard** est utilisé, toutes les modifications sont ignorées.

```
$ git revert [commit sha]
```

Crée un nouveau commit, annulant les modifications du commit spécifié. Elle génère une **inversion** des changements.

08 Synchronisation des référentiels

```
$ git fetch [remote]
```

Récupérez les modifications à partir du **remote**, mais ne mettez pas à jour les branches de suivi.

```
$ git fetch --prune [remote]
```

Supprimez les références distantes qui ont été supprimées du référentiel **remote.**

```
$ git pull [remote]
```

Récupérer les modifications de la branche **remote** et fusionner avec sa branche en amont.

```
$ git push [--tags] [remote]
```

Poussez les modifications locales vers le **remote**. Utilisez **--tags** pour pousser les balises.

```
$ git push -u [remote] [branch]
```

Poussez la branche locale vers le référentiel **remote** . Définissez sa copie en amont.

Commit	un objet
Branch	**une référence à un commit ; peut avoir un suivi en amont**
Tag	une référence (standard) ou un objet (annoté)
Head	un endroit où se trouve maintenant **votre répertoire de travail**

A Git installation

Pour les distributions GNU/Linux, Git doit être disponible dans le référentiel système standard. Par exemple, dans Debian/Ubuntu, veuillez saisir dans le **terminal**:

```
$ sudo apt-get install git
```

Si vous avez besoin d'installer Git à partir de la source, vous pouvez l'obtenir à partir de git-scm.com/do wnloads.

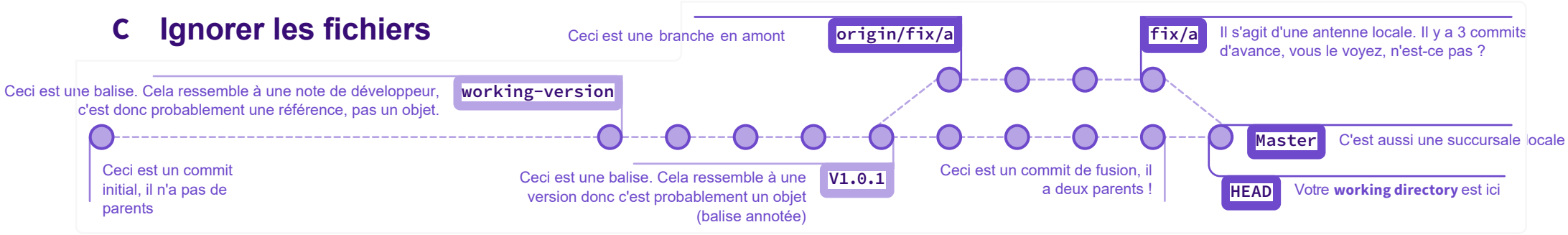
Un excellent cours Git peut être trouvé dans le grand livre **Pro Git** de Scott Chacon et Ben Straub.

B Ignorer les fichiers

```
$ cat .gitignore
/logs/*
!logs/.gitkeep
/tmp
*.swp
```

Vérifiez que le fichier .gitignore existe dans votre projet et ignorez certains types de fichiers, tels que tous les fichiers du répertoire des **logs** (à l'exception du fichier **.gitkeep**), tout le répertoire **tmp** et tous les fichiers ***.swp**. L'ignorance des fichiers fonctionnera pour le répertoire (et les répertoires enfants) où le fichier **.gitignore** est placé.

C Ignorer les fichiers



D Le zoo des zones de travail