Support TP de la formation « Commencer Git »[1]

1 Description du cas d'utilisation

Dans l'exemple de ce TP, nous étudierons la création d'un dépôt destiné à recueillir un listing de vos poésies préférées (dans un format texte simple). Un dépôt est déjà existant et contient quelques poésies : on se donne pour tâche, d'une part, de corriger les poésies déjà présentes et, d'autre part, de compléter la collection.

Le dépôt est cloné sur votre ordinateur avec[2]

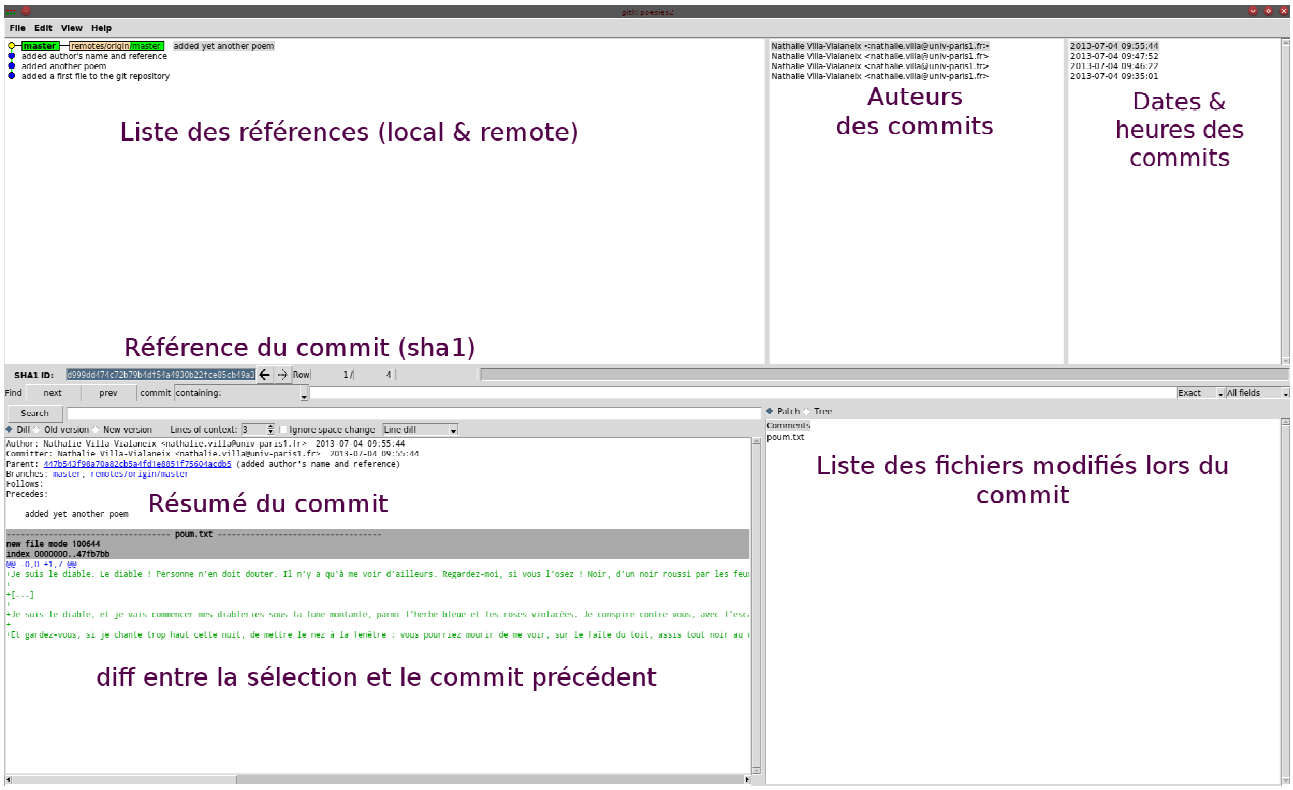
```
git clone https://github.com/tuxette/tuto.git poesies
```

Le présent dépôt est accessible en lecture à tous (mais pas en écriture...).

Entrer dans le dépôt local et examiner son contenu :

```
cd poesies
ls
gitk &
```

gitk est une interface utilisateur graphique permettant de visualiser un dépôt Git. Il existe d'autres interfaces de ce type : git-gui, github (sur le web) ou bien RStudio (pour les utilisateurs de R), etc



Dans gitk :

1. *View/New view* ; remplir :

---

1 De manière alternative, si cette URL ne fonctionnait pas : git clone http://git.nathalievilla.org/git-tuto.git poesies

1. *View name* : all
2. ***Remember this view***
3. ***All refs*** / ***All (local) branches*** / ***All tags*** / ***All remote tracking branches***

2. *View*/*New view* ; remplir :
   1. *View name* : poum
   2. ***Remember this view***
   3. ***Enter files and directories to include, one per line*** : poum.txt

2 Configuration

Lors de la première configuration, il faut configurer son profil utilisateur, a minima :

```
git config --global user.name "Tuxette Chix"
git config --global user.email tuxette@chix.nathalievilla.org
git config --global core.editor nano
```

On vérifie la configuration globale avec

```
git config –list
```

3 Manipulations de base pour travailler son dépôt local (commit / tag / add / mv / rm)

On vérifie le statut d'un dépôt local avec la commande

```
git status
```

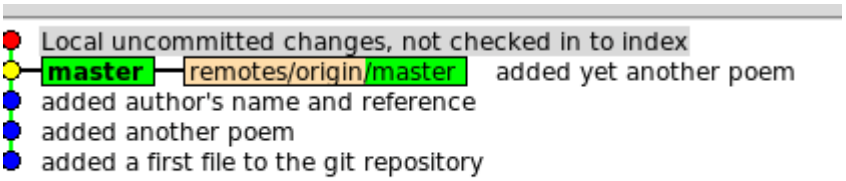
Ajouter[3] une poésie (appelée **mignonne.txt** dans la suite) de votre choix dans le dossier et faire de nouveau

```
git status
```

Éditer le fichier **dormeur.txt** et en fin de fichier ajouter une ligne et le texte : **Appréciation :** *** puis faire :

```
git status
git diff
```

et regarder gitk (touche ***F5***) :



On commit localement la modification par :

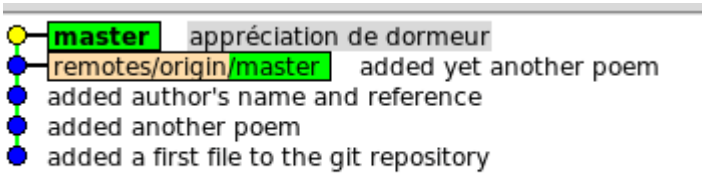
```
git commit dormeur.txt
```

Entrer le message de commit **appréciation de dormeur** puis faire :

3 Normalement, en phase de développement, on ne modifie pas directement dans la branche « master » ; pour simplifier le TP, nous commencerons néanmoins par cela.

git status

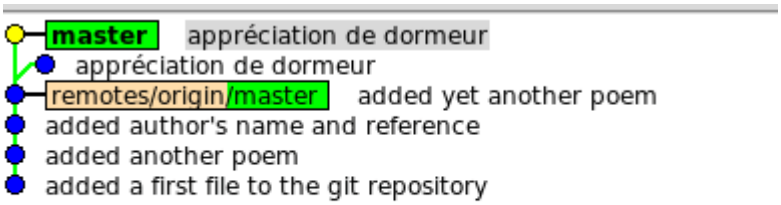
et regarder gitk (touche *F5*) :



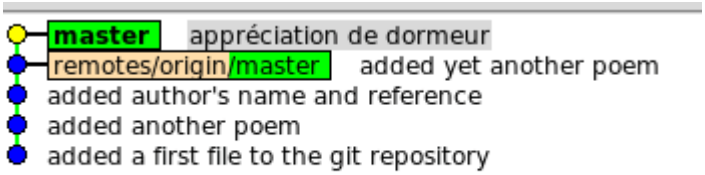
Vous souhaitez ajouter votre nom à la fin de l'appréciation : éditez le fichier dormeur.txt et modifier la ligne **Appréciation : ***** en **Appréciation : *** (NV2)** (utilisez votre propres initiales). Plutôt que de faire un nouveau commit, on « amend » le précédent avec

```
git commit --amend dormeur.txt
git status
```

et regarder gitk (touche *F5*) :



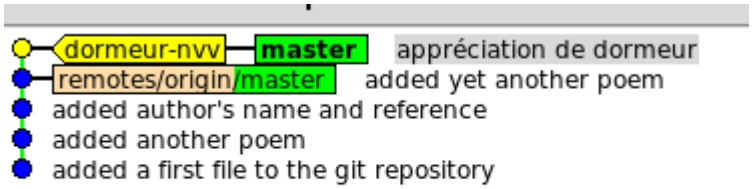
puis touches *Ctrl+F5* :



Si on souhaite repérer cette référence de manière plus simple qu'avec le sha1, on peut ajouter un tag :

```
git tag dormeur-nvv
```

et regarder gitk (touche *F5*) :



Ajouter la poésie au dépôt (pour les besoins du TP, n'appelez pas votre fichier de la même manière que moi) :

```
git add mignonne.txt
git status
```

puis commiter la modification et ajouter un tag (à personnaliser) :

```
git commit –a -m "ajout de mignonne"
git tag  "mignonne"
```

puis modifier le nom du fichier

```
git mv mignonne.txt autre.txt
git commit –a -m "modification du nom de mignonne"
git status
```

avant de décider que, finalement, ce fichier n'est vraiment pas intéressant

```
git rm autre.txt
git commit –a -m "je vire mignonne"
git status
ls
```

Regarder et regarder gitk (touche ***F5***). Pris d'un dernier regret, vous souhaitez revenir à l'état de l'ajout de **mignonne.txt**

```
git reset --hard "mignonne"
ls
```

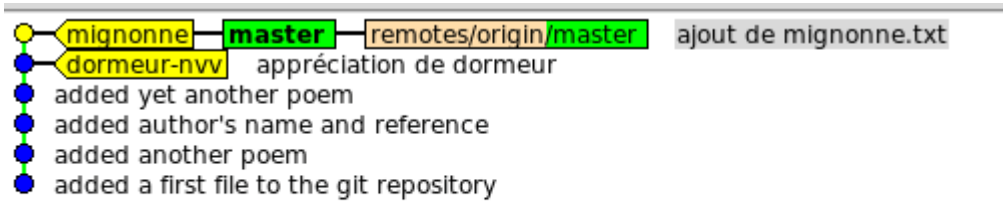
puis regarder gitk (touche ***F5*** puis touches ***Ctrl+F5***).

4 Manipulations de base pour travailler avec un dépôt distant (push / pull / blame)

À ce stade, je pousse mes modifications sur le serveur (ce que vous ne pouvez pas faire car vous n'êtes pas enregistrés comme collaborateurs) :

```
git push
```

gitk (touche ***F5*** donne ceci, pour moi) :



Récupérez mes modifications avec un

```
git pull
```

Ici des conflits doivent apparaître (conflit au niveau du fichier dormeur.txt, modifié différemment par chacun d'entre nous) : ouvrir ce fichier et résoudre le conflit puis faire

```
git commit –a -m "résolution du conflit"
```

On peut facilement retrouver l'auteur et la date d'un bug introduit dans les codes grâce à la commande :

```
git blame dormeur.txt
```

Enfin, on peut se déplacer dans l'historique du dépôt avec

```
git checkout dormeur-nvv
ls
git checkout master
ls
```

5 Les branches

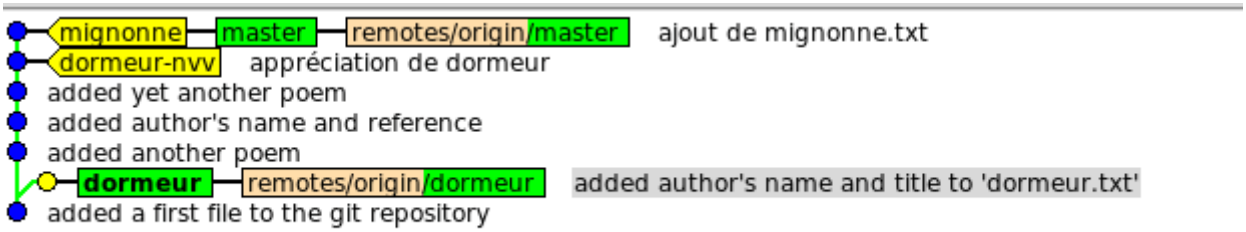
Notre dépôt contient deux branches : **master** et **dormeur** (sur laquelle on s'est occupé de la mise en forme du fichier **dormeur.txt**). On liste les branches avec :

```
git branch --all
```

On bascule sur la branche **dormeur** avec :

```
git checkout dormeur
ls
```

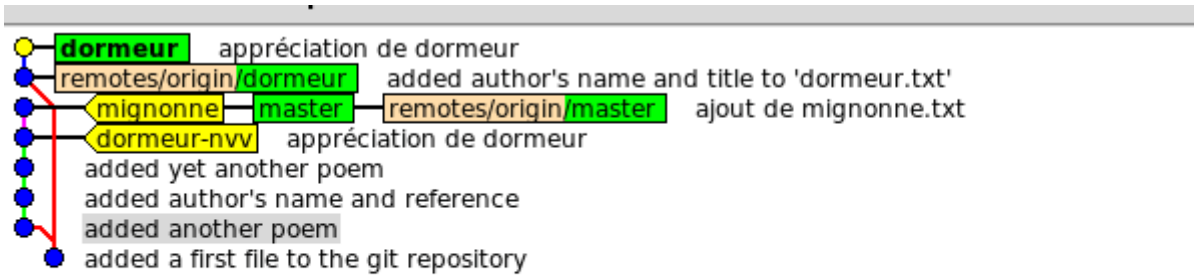
puis on regarde gitk (touche ***F5***)[4] :



On peut aller ajouter à cette branche, une modification faite dans une autre branche (sans fusionner) avec la commande :

```
git cherry-pick dormeur-nvv
```

qui ajoute dans la branche **dormeur** l'appréciation introduite entre le commit **dormeur-nvv** et la référence précédente de la branche **master**. On visualise la différence avec gitk (touche ***F5***) :



Une fois que la branche de développement **dormeur** est satisfaisante, on peut revenir dans la branche master pour fusion (on fusion la branche **dormeur** DANS la branche **master** et pas l'inverse) :

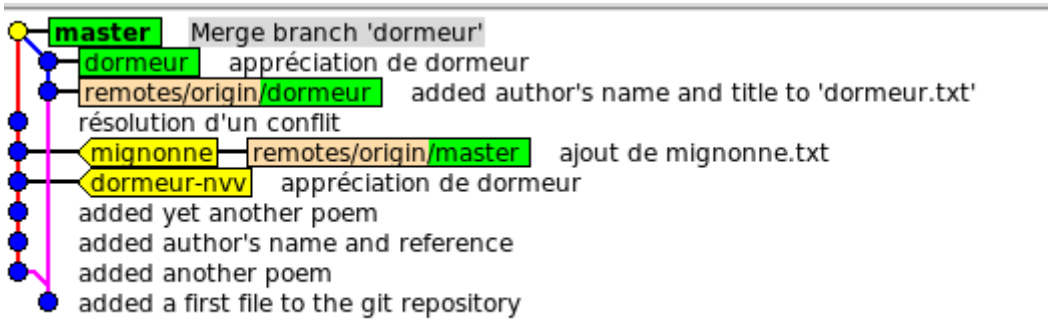
```
git checkout master
git merge dormeur
```

qui doit résulter en un conflit au niveau de l'appréciation (à nouveau le même que celui que vous

4 L'utilisation de ***Ctrl+F5*** fera disparaître la référence non utilisée, ici **master**.

avez eu avec la commande pull). On peut dire directement à Git de favoriser notre version lors de la fusion :

```
git reset --hard HEAD
git merge-recursive ours dormeur
```



On peut créer une nouvelle branche, à l'emplacement courant (que l'on définit éventuellement avec la commande checkout) :

```
git branch "auteurs"
```

pour faire une branche où l'on va ajouter les auteurs. On se déplace dans cette branche avec

```
git checkout auteurs
```

puis on ajoute des noms d'auteurs et on finit avec :

```
git commit -a -m "ajout des noms d'auteurs"
git checkout master
git merge auteurs
```

6 En savoir plus...

Référence : https://help.github.com

Personne à **ne pas** prendre en référence : NV² 

La preuve en image : http://youtu.be/sZSaIf6in3g ! ;-)

Créer / héberger un dépôt Git

Github https://github.com permet de créer de manière simple des dépôts Git et d'y ajouter des utilisateurs. Github dispose aussi d'une interface graphique de suivi des projets.

Attention, dans la version gratuite de Github, on ne peut créer que des dépôts publics (i.e., lisibles par tous). Pour des dépôts privés, il faut soit passer par un abonnement, soit disposer de son propre serveur Git. Installation d'un serveur Git : http://tuxette.nathalievilla.org/?p=405 (gitolise, anciennes version d'Ubuntu Server) et http://tuxette.nathalievilla.org/?p=780 (gitolite, nouvelles version d'Ubuntu Server).

Configuration locale

Chaque dépôt local peut aussi avoir sa propre configuration spécifique que l'on définit dans le fichier **.git/config**. Configuration avancée de git :

http://git-scm.com/book/en/Customizing-Git-Git-Configuration

On peut aussi définir une liste de fichiers à ne pas inclure dans le dépôt (utile notamment si on ne veut pas avoir des sorties illisibles dans git status) en éditant le fichier **.gitignore**.

Modification a posteriori des commits, options de reset et références de checkout

Il est possible de retravailler complètement a posteriori ses commits (fusion / split / réécriture des messages...) avec l'option

```
git rebase -i REF
```

Voir http://git-scm.com/book/fr/Les-branches-avec-Git-Rebaser pour plus d'informations sur cette fonctionnalité que je ne maîtrise pas du tout.
La commande **reset** dispose de plusieurs options dont les plus utiles (pour moi) sont :

- **--hard** : supprime les références de commit et les modifications sur les fichiers ;
- **--soft** : supprime les références de commit mais pas les modifications sur les fichiers.

De plus, alternativement à l'utilisation du tag, on peut utiliser la référence par le sha1 ; dans mon exemple, cela aurait donné :

```
git reset --hard 0f697854df341c99ebb3299bfe00684be8c6ecdd
```

On peut utiliser **checkout** avec en référence le nom d'une branche (on remonte alors à la tête de la branche), un tag ou une référence sha1 pour se déplacer dans l'historique du dépôt :

```
git checkout master
git checkout dormeur-nvv
git checkout 0f697854df341c99ebb3299bfe00684be8c6ecdd
```

Les stratégies pour merge

Quelques stratégies utiles pour merge :

```
git merge -s ours
git merge -s recursive
git merge-recursive
git merge-recursive ours
git merge-recursive theirs
```

Plus d'informations avec man **git-merge**.