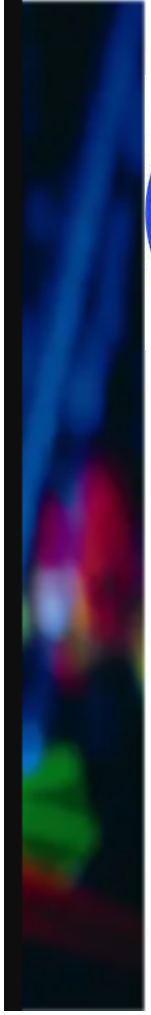

Université de N'Djamena

Data Developper

Module : GIT et GITHUB

THÈME :

Présentation des commandes GIT

présenté par

ALLARABAYE Julie

LAGRE GABBA Bertrand

MAHAMAT Saleh Yacoub
MBAYAM Hippolyte Djegomdé

le 20 janvier 2023

Enseignant principal : **M. SAKAYO TOUADOUM**

Année 2022-2023

ACTIVITE

Table des matières

Table des figures

INTRODUCTION GÉNÉRALE

L'évolution technologique a connu un essor fulgurant et une ampleur majeure. Elle a fait naître de grands projets. Des applications, des sites web et des plateformes ont vu le jour. Ces réalisations sont le fruit du travail des concepteurs, développeurs et analystes. De part la taille du projet, lon peut constituer une équipe. Mais un certain nombre de questions se pose : **comment faire pour garder l'historique de l'évolution de chaque module ? Comment peut-on faire pour collaborer avec d'autres personnes sur le même sujet ? Quel outil utilisé ?** Ces questions ont conduit à la naissance de l'outil **git** dont certaines commandes seront présentées afin de les maîtriser.

Chapitre 1

PRÉSENTATION DE GIT ET GITHUB

1.1 Définition et Historique

Git est un logiciel de versioning créé en 2005 par Linux Torvalds. Un logiciel de versioning, ou logiciel de gestion de version est un logiciel qui permet de conserver un historique des modifications effectuées sur un projet afin de pouvoir rapidement identifier les changements effectués et de revenir à une ancienne version en cas de problème. Les logiciels de gestion de versions sont quasiment incontournables aujourd'hui car ils facilitent grandement la gestion de projets et permettent de travailler en équipe de manière beaucoup plus efficace. Parmi les logiciels de gestion de versions, **Git** est le leader incontesté et il est donc indispensable pour tout développeur de savoir utiliser Git. Ainsi, notre travail porte sur quelques commandes de git indispensables pour tout développeur.

FIGURE 1.1 – git image

Chapitre 2

RÔLE DES DIFFERENTES COMMANDES GIT

2.1 git annotate

La commande **annotate** est utilisée dans git pour suivre chaque ligne du fichier en fonction des informations de validation. Cette commande annote à partir de la révision donnée du fichier.

2.1.1 Exemple 1 git annotate

```
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/git_learning/src$ git annotate index.py
00000000        (Not Committed Yet      2023-01-19 12:32:21 +0100       1)print("version4")
32419573        (  FoubaDev     2023-01-17 17:07:24 +0100       2)print("this file is the body of our project")
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/git_learning/src$
```

FIGURE 2.1 – git annotate

2.1.2 Exemple 2 git annotate avec ”e” pour afficher l’adresse mail

```
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/git_learning/src$ git annotate -e  index.py
00000000        (<not.committed.yet>    2023-01-19 12:38:34 +0100       1)print("version4")
32419573        (<foubabertrand@gmail.com>      2023-01-17 17:07:24 +0100       2)print("this file is the body of our proje
ct")
```

FIGURE 2.2 – Affichage de l’adresse mail de l’auteur avec git annotate

2.1.3 Exemple 3 git annotate pour afficher l’heure

```
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/git_learning/tkinter_project$ git annotate -t database.py
905aa1f6        (  FoubaDev     1674128711 +0100        1)# this file is used to configure the connection with the database
905aa1f6        (  FoubaDev     1674128711 +0100        2)
905aa1f6        (  FoubaDev     1674128711 +0100        3)# we will use postgresql to keep all data coming from form
```

FIGURE 2.3 – git annotate pour l’heure

2.2 git blame

La commande **git blame** est un utilitaire de dépannage doté d’options d’utilisation étendues. La fonction de haut niveau de git blame est l’affichage des métadonnées de l’auteur. Ceci est utilisé pour examiner des points spécifiques de l’historique d’un fichier et obtenir un contexte sur le dernier auteur qui a modifié la ligne. Ceci est utilisé pour explorer l’historique d’un code spécifique et répondre aux questions sur **quoi**, **comment** et **pourquoi** le code a été ajouté à un référentiel.

2.2.1 Exemple 1 git blame

```
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/git_learning$ git blame config.py
5ae94fa9 (FoubaDev 2023-01-17 14:30:25 +0100 1) print("Help for project3")
5ae94fa9 (FoubaDev 2023-01-17 14:30:25 +0100 2) print("hello,world3")
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/git_learning$
```

FIGURE 2.4 – git blame

2.2.2 Exemple 2 git blame

```
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/groupe1_project$ git blame README.md
^cc647c9 (FoubaDev 2023-01-16 16:33:34 +0100 1) c'est un projet de groupe1 portant sur le travail collaboratif github
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/groupe1_project$
```

FIGURE 2.5 – git blame exemple 2

2.3 git log

La liste de tous les **commit**s dans le référentiel git est récupérée à l'aide de la commande **git log**. Elle affiche les détails de chaque commit tels que : le message associé au commit.

2.3.1 Exemple git log

```
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/git_learning$ git log
commit 324195732b4571192452cd3c035406d38faec5ca (HEAD, tag: v2.0, tag: v1.0)
Author: FoubaDev <foubabertrand@gmail.com>
Date:   Tue Jan 17 17:07:24 2023 +0100

    version4

commit 4b0c7beec7abe7f4bcbcf181a8e90b26c52ed991
Author: FoubaDev <foubabertrand@gmail.com>
Date:   Tue Jan 17 16:04:10 2023 +0100

    version1

commit 05a638d6a20a3037cb4d1b9a833d23c77cbfba6f (origin/master)
Author: FoubaDev <foubabertrand@gmail.com>
Date:   Tue Jan 17 13:26:23 2023 +0100

    add some line in the file

commit 4b8d99c6fe0f2171fdb0960c61652474ba46184c
Author: FoubaDev <foubabertrand@gmail.com>
Date:   Tue Jan 17 13:18:46 2023 +0100

:...skipping...
commit 324195732b4571192452cd3c035406d38faec5ca (HEAD, tag: v2.0, tag: v1.0)
```

FIGURE 2.6 – git log

On peut afficher les deux dernières modifications avec loption **-n 2**.

2.3.2 Exemple 2 git log

```
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/git_learning$ git log -n 2
commit 324195732b4571192452cd3c035406d38faec5ca (HEAD, tag: v2.0, tag: v1.0)
Author: FoubaDev <foubabertrand@gmail.com>
Date:   Tue Jan 17 17:07:24 2023 +0100

    version4

commit 4b0c7beec7abe7f4bcbcf181a8e90b26c52ed991
Author: FoubaDev <foubabertrand@gmail.com>
Date:   Tue Jan 17 16:04:10 2023 +0100

    version1
```

FIGURE 2.7 – git log exemple 2

2.4 git diff

Il fournit des informations détaillées sur les mises à jour effectuées.

2.4.1 Exemple git diff

```
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/git_learning$ git diff
diff --git a/src/index.py b/src/index.py
index b5e8be6..1869d37 100644
--- a/src/index.py
+++ b/src/index.py
@@ -1,2 +1,2 @@
-print("version2")
+print("version4")
 print("this file is the body of our project")
\ No newline at end of file
```

FIGURE 2.8 – git diff

2.5 git reset

Il est utilisé pour réinitialiser la zone **staging** en gardant le répertoire du référentiel intact. **reset** est la commande que nous utilisons lorsque nous voulons déplacer le référentiel vers une validation précédente, en ignorant toutes les modifications apportées après cette validation.

2.6 git restore

La commande **restore** permet d'annuler les modifications locales non validées

2.6.1 Exemple git restore

```
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/git_learning/src$ git restore --staged reset3
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/git_learning/src$
```

FIGURE 2.9 – git restore

2.7 git –amend -m ”le message”

Si le commit existe uniquement dans le référentiel local et n'a pas été poussé sur GitHub.com, on peut corriger le message de commit avec la commande **git commit –amend**.

2.7.1 Exemple git –amend -m ”le message

```
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/git_learning/src$ git commit --amend -m "tech4tchad commit"
[master 5ae94fa] tech4tchad commit
 Date: Tue Jan 17 14:30:25 2023 +0100
 2 files changed, 3 insertions(+), 2 deletions(-)
 create mode 100644 src/reset3
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/git_learning/src$
```

FIGURE 2.10 – git –amend -m ”le message

2.8 git revert

Cette commande est utilisée pour annuler les modifications d'un seul commit. L'historique de git reste inchangé après l'exécution de cette commande qui est importante pour le travail d'équipe git revert s'execute après **git reflog**.

2.8.1 Exemple git revert

2.9 git reflog

Cette commande est utilisée pour afficher les enregistrements des mises à jour effectuées sur différentes branches du référentiel. Il fournit des informations sur l'état précédent des branches qui aident l'utilisateur git a revenir à l'état précédent si nécessaire.

```
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/groupe1_project$ git reflog
f24bb85 (HEAD -> master, origin/master) HEAD@{0}: reset: moving to f24bb85
f24bb85 (HEAD -> master, origin/master) HEAD@{1}: commit: ajout
c202760 HEAD@{2}: revert: Revert "Revert "version1 js""
f2d2f17 HEAD@{3}: revert: Revert "version1 js"
53c9155 HEAD@{4}: reset: moving to HEAD
53c9155 HEAD@{5}: reset: moving to HEAD
53c9155 HEAD@{6}: commit: version1 js
5ac0c65 HEAD@{7}: pull --tags origin master: Fast-forward
cc647c9 (origin/Julie, Julie) HEAD@{8}: checkout: moving from Julie to master
cc647c9 (origin/Julie, Julie) HEAD@{9}: reset: moving to HEAD
cc647c9 (origin/Julie, Julie) HEAD@{10}: checkout: moving from master to Julie
cc647c9 (origin/Julie, Julie) HEAD@{11}: commit (initial): ajout de readme.md
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/groupe1_project$ git revert f24bb85
Suppression de js/register.js
Suppression de js/admin.py
[master 87528e9] Revert "ajout"
 2 files changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 js/admin.py
 delete mode 100644 js/register.js
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/groupe1_project$
```

FIGURE 2.11 – git revert

```
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/groupe1_project/js$ git reflog
53c9155 (HEAD -> master, origin/master) HEAD@{0}: reset: moving to HEAD
53c9155 (HEAD -> master, origin/master) HEAD@{1}: reset: moving to HEAD
53c9155 (HEAD -> master, origin/master) HEAD@{2}: commit: version1 js
5ac0c65 HEAD@{3}: pull --tags origin master: Fast-forward
cc647c9 (origin/Julie, Julie) HEAD@{4}: checkout: moving from Julie to master
cc647c9 (origin/Julie, Julie) HEAD@{5}: reset: moving to HEAD
cc647c9 (origin/Julie, Julie) HEAD@{6}: checkout: moving from master to Julie
cc647c9 (origin/Julie, Julie) HEAD@{7}: commit (initial): ajout de readme.md
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/groupe1_project/js$
```

FIGURE 2.12 – git reflog

2.9.1 Exemple git reflog

2.10 git reset HEAD-2

Les branches Git sont très utiles, et vous pouvez créer une nouvelle branche, fusionner une branche ou supprimer une branche, selon vos besoins. Il existe de nombreuses commandes git que vous pouvez utiliser pour gérer vos branches dans git.

Lorsqu'on utilise la branche git checkout, HEAD indique le dernier commit. En termes simples, on peut dire que Git HEAD est la branche actuelle. Chaque fois qu'on extrait une branche ou crée une nouvelle branche, Git HEAD la transfère.

HEAD est une référence au dernier commit dans la branche d'extraction actuelle.

2.10.1 Exemple git reset HEAD-2

```
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/groupe1_project$ git reset HEAD~2
Modifications non indexées après reset :
M       login.py
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/groupe1_project$
```

FIGURE 2.13 – git reset HEAD-2

2.11 git rebase

Cette commande fonctionne comme la commande **git merge**. La différence est qu'il est utilisé pour fusionner les commits d'une branche dans une autre branche un par un.

2.11.1 Exemple git rebase

```
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/groupe1_project$ git rebase master
Rembobinage préalable de head pour pouvoir rejouer votre travail par-dessus...
Avance rapide de Julie sur master.
tech4tchad13@tech4tchad13:~/Documents/Tech4Tchad/GitHub/groupe1_project$
```

FIGURE 2.14 – git rebase

Conclusion

Pour pouvoir garder lhistorique de son projet, Git est incontournable. Non seulement il est un logiciel de versioning (c'est-à-dire d'historique de versions), il permet de travailler aussi en équipe. Parmi les logiciels de gestion de versions, il fait partie des leaders incontestés. Tout développeur souhaitant évoluer dans de grands projets et collaborer avec d'autres développeur est appelé à utiliser git grâce à différentes commandes disponibles.

Bibliographie

[1] Suppourt de l'enseignant principal, SAKAYO TOUADOUM.

[2] Fiche Travail de groupe de l'enseignant principal

Webographie

[site01] https://git-scm.com/docs/git-reset **consulté le 19/01/2023**

[site02] https://stackoverflow.com/questions/13321556/difference-between-git-and-github **consulté le 19/01/2023**

[site03] https://git-scm.com/book/en/v2/Getting-Started-The-Command-Line **consulté le 19/01/2023**

[site04] https://docs.github.com/en/actions **Consulté le 20/01/2023**