

Cours : LOG1000

Rapport Travail pratique 1: GIT et Makefiles.

École polytechnique de Montréal

**Trimestre :** automne 2018

Équipier1: CYRILLE TALLA FONGANG

Équipier2: GEORGES PARFAIT *DJIMEFO KAPEN*

**Équipe: 04**

**Présenté à :**

Polytechnique Montréal

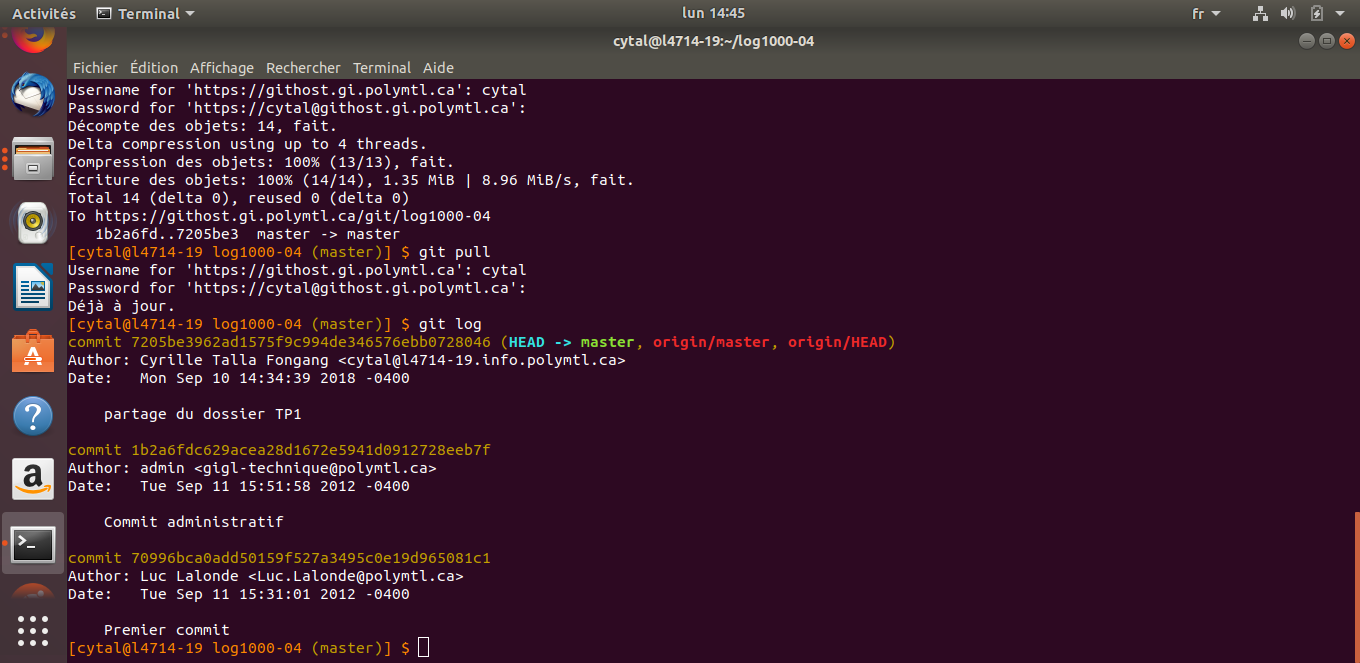
date de remise(10-09-2018)

Travail pratique #1: GIT et Makefiles

**Partie** **1 : GIT**

***E1.1 Mise en place du répertoire GIT***

a) capture d’écran de la commande de la commande « git log »

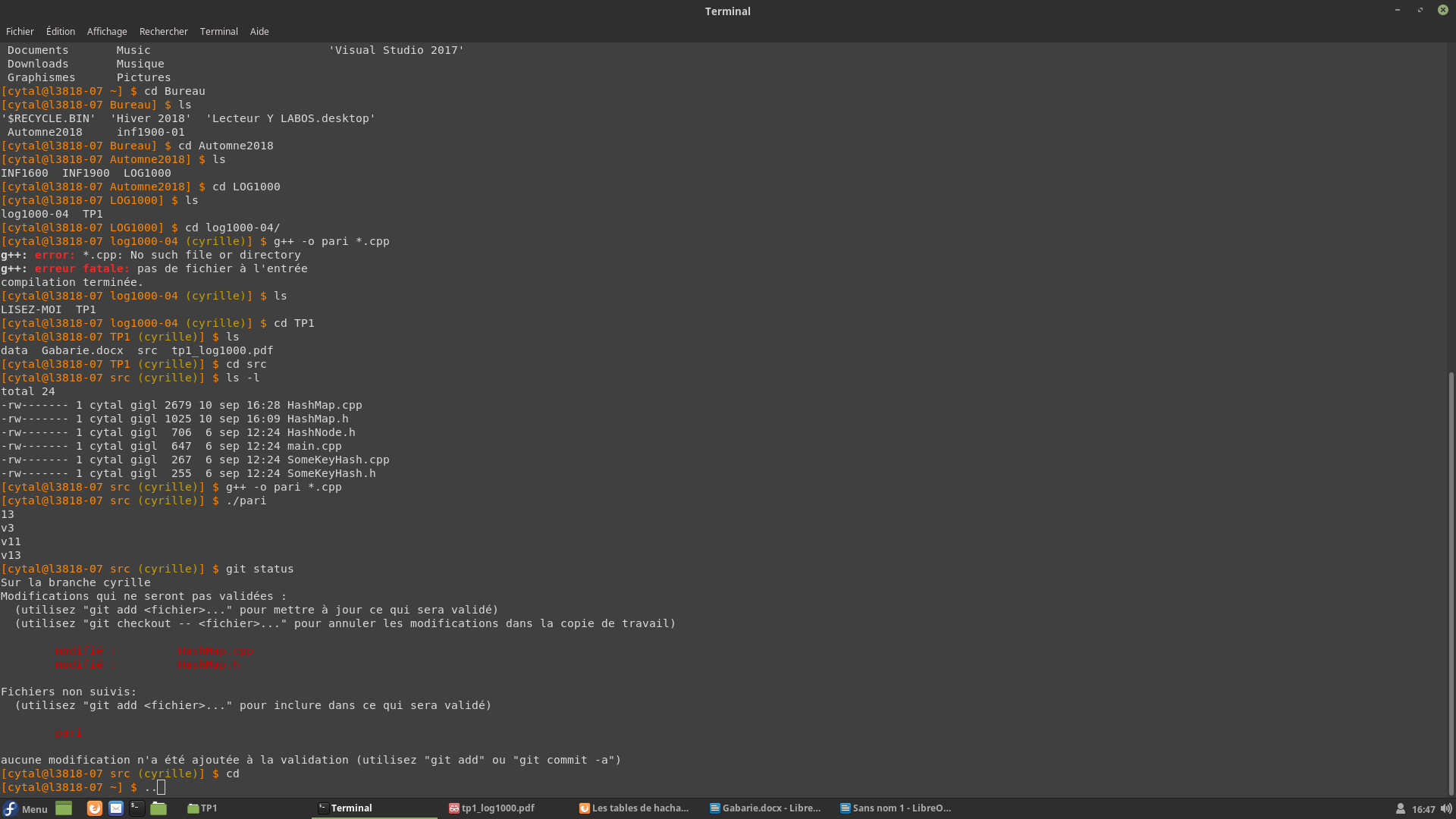


b) Différence entre les commandes « git log » et « git log -p»

La commande « git *log* » permet d’obtenir en ordre chronologique les commits effectués. Tandis que la commande « git *log - p*» permet d’avoir le détail des lignes qui ont été ajoutées et retirées dans chaque commit [1].

***E1.2 Modification 1***

3-a) Écran de sortie

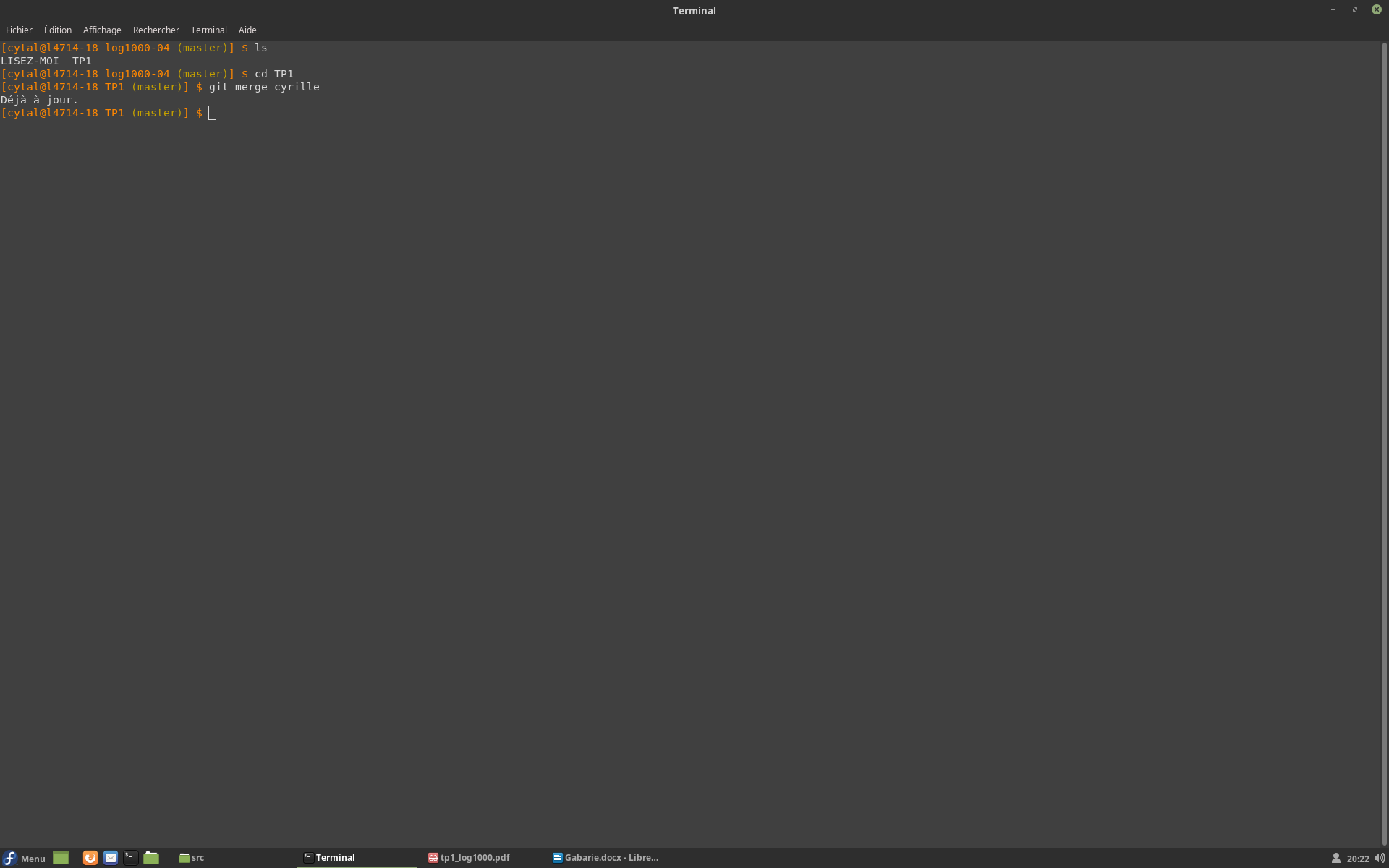


3-b) Est ce que cette situation est normale?

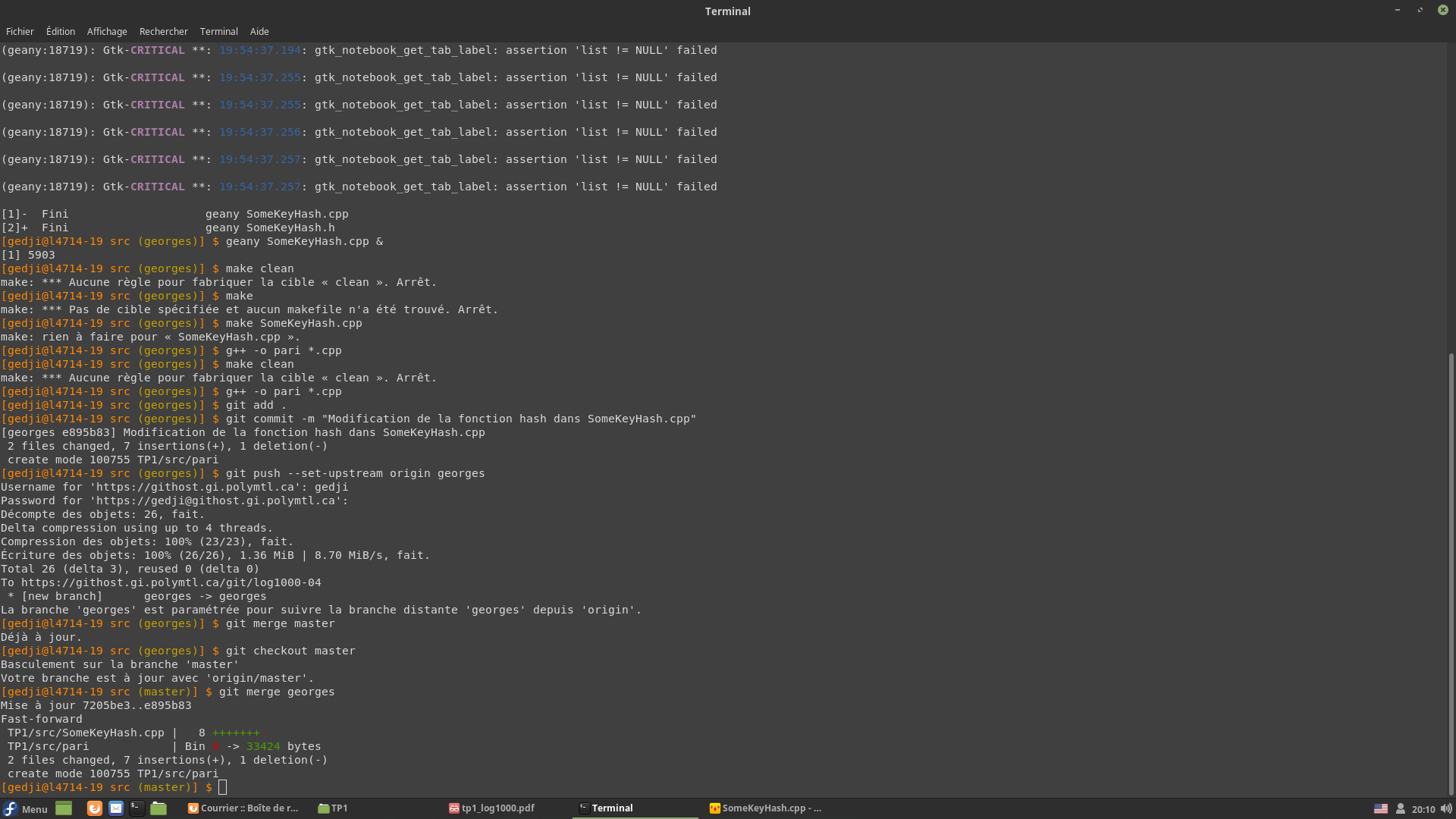
Cette situation est normale. En effet, les modifications apportées aux fichiers « HashMap.h » et « HashMap.cpp » ne sont effectives que dans le working directory en attendant d’être ajoutées (validées) dans le staging area (avec la commande git add) et soumises dans le répertoire local (avec la commande git commit).

5-a) Captures d’écran des deux équipiers correspondantes a l’exécution de la commande « git *merge* »

Équipier 1:



Équipier 2:



5-b) Détection de conflit

Git ne détecte aucun conflit pour la simple raison que les deux équipiers ont travaillé sur 2 branches différentes et 2 fichiers distincts avant de faire un push vers le serveur.

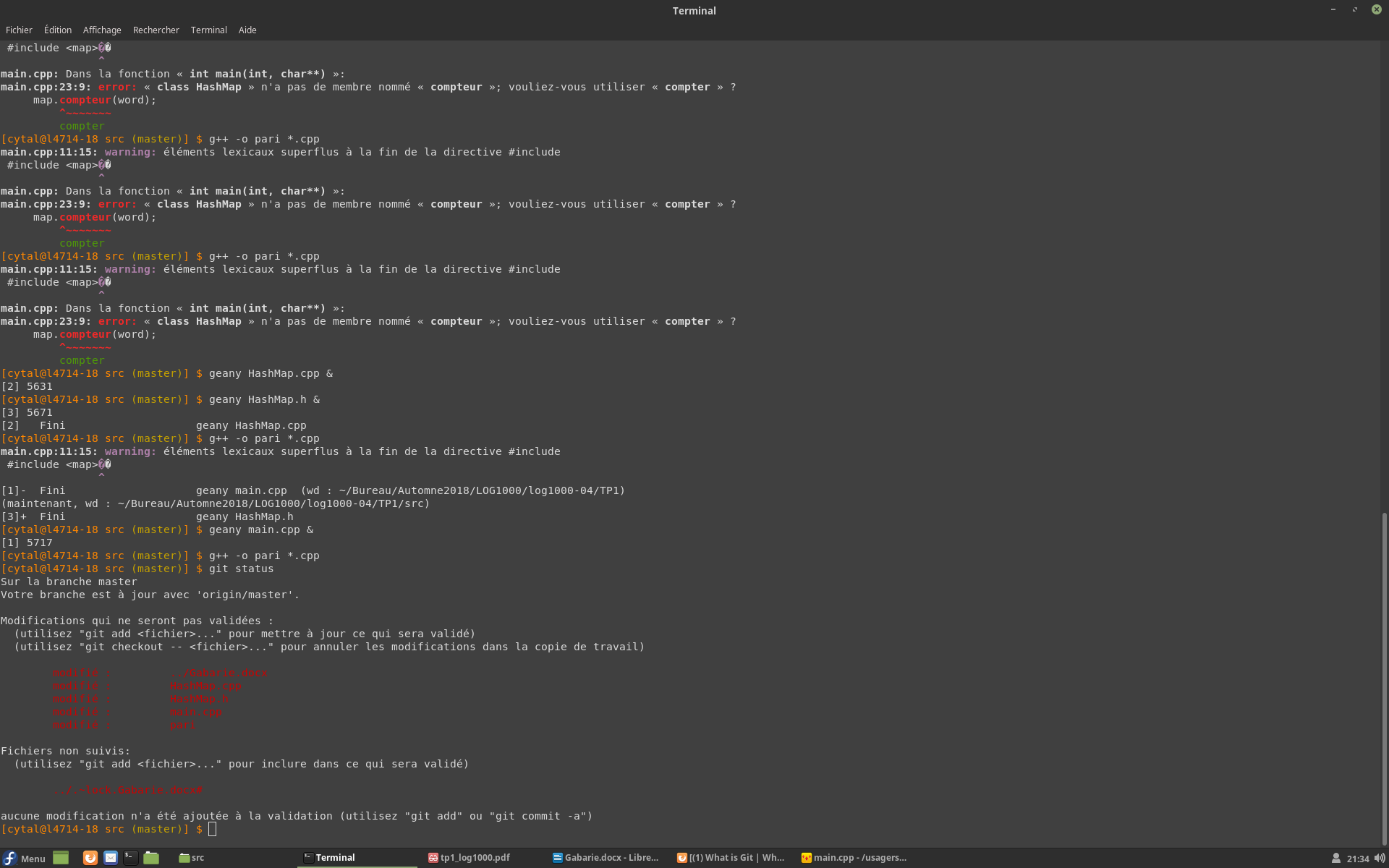
Il ne nous est pas possible de fournir dans ce gabarit la capture d’écran de sortie du LOG, car dans le document de référence du TP (TP1\_LOG1000.pdf) ne figurait pas cette instruction. C’est a la question 5-d de la partie E1.4 que le document de référence fait mention de cette capture pour la première fois.

***E1.3 Modification 2***

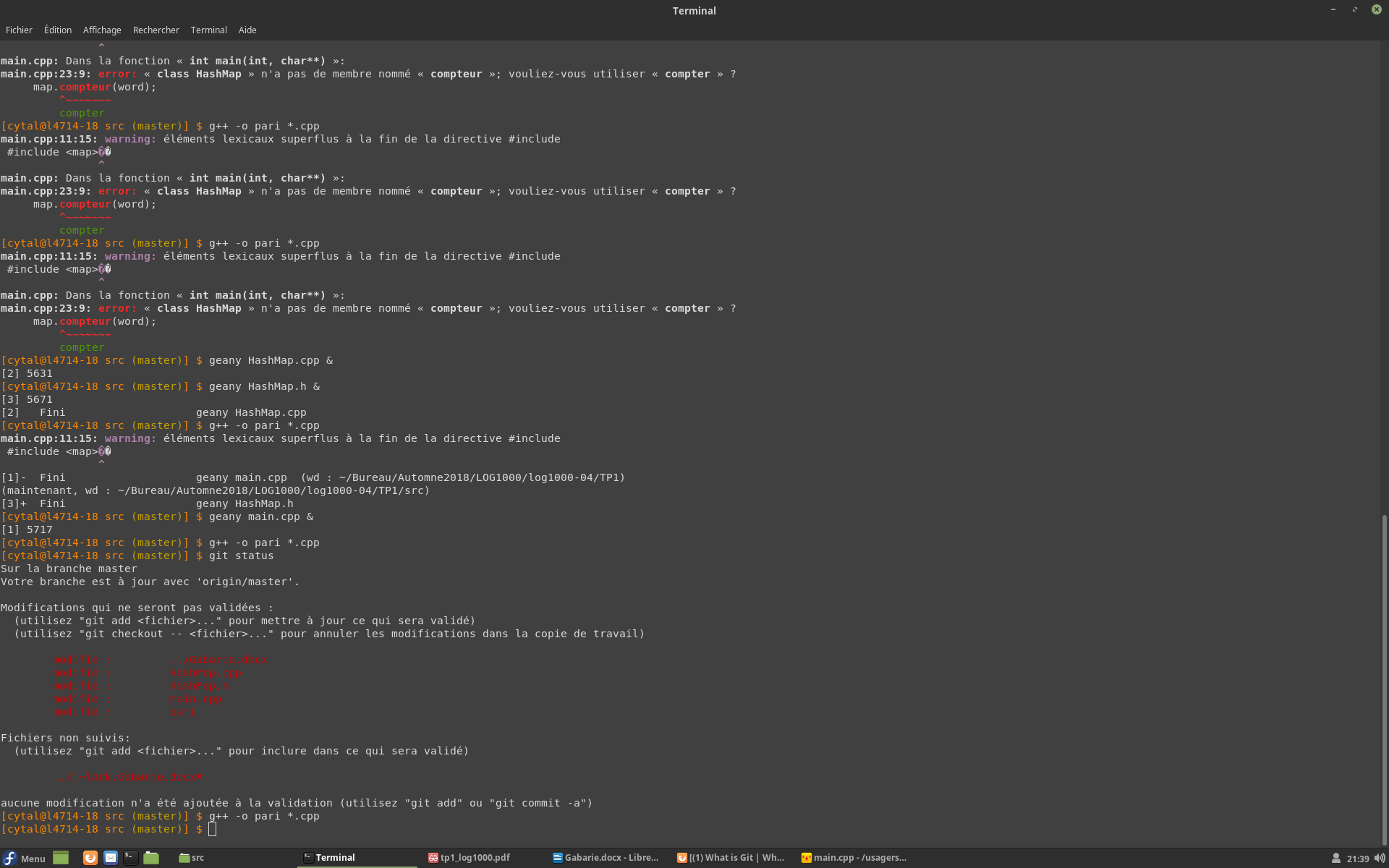
*3-a) Pourquoi le nom de l'exécutable, HashMap.h et HashMap.cpp sont écris en rouge dans la console ?*

Les noms de l’exécutable et des fichiers main, HashMap.h et HashMap.cpp sont écrits en rouge sur la console car il ont été *modifiés.*

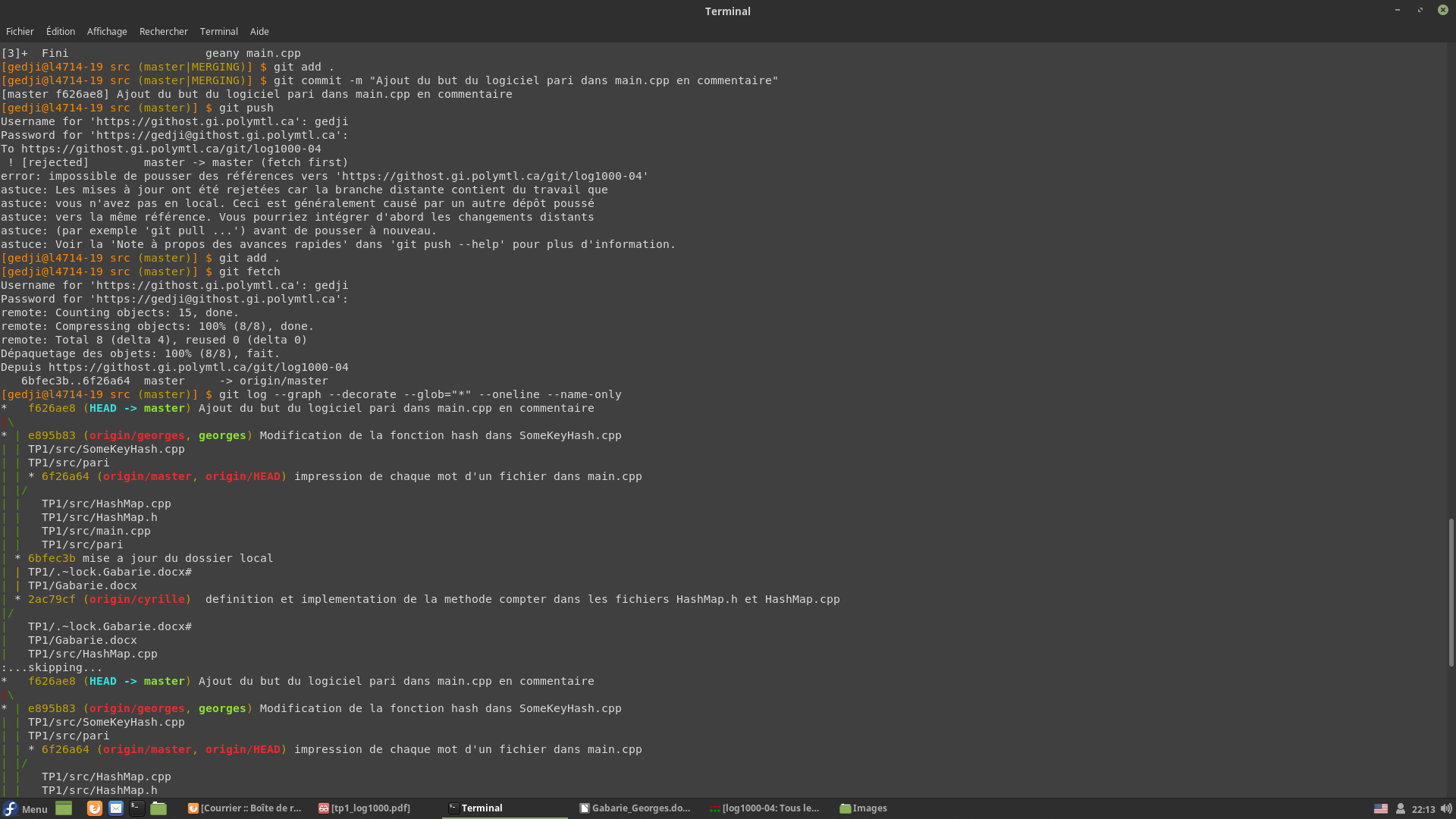
*3-b) Capture d’écran de la sortie du terminal correspondante à l'exécution de la commande.*

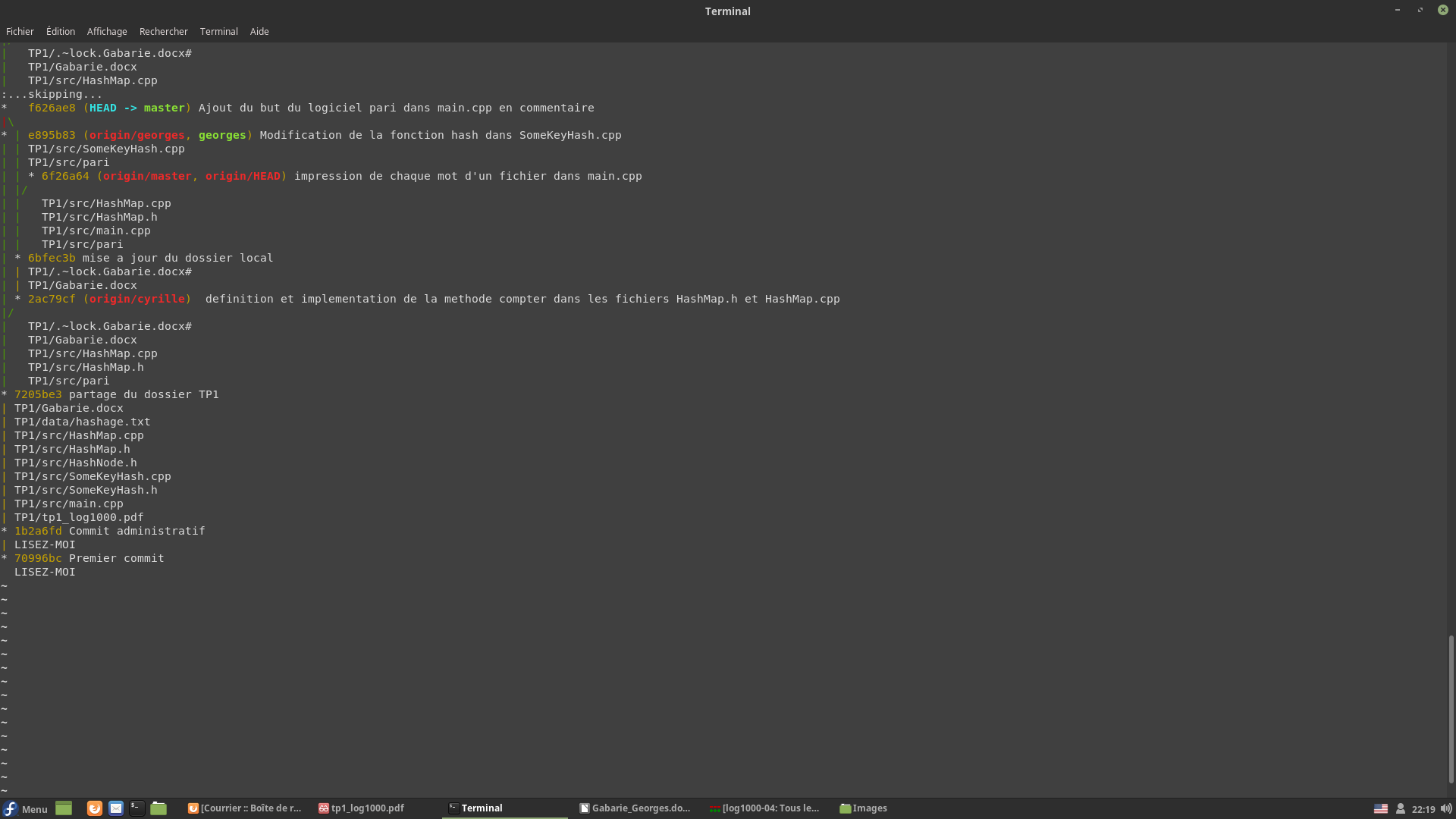


*3-c) Compilation du code source et capture d’écran de la sortie du programme.*



5-a) Capture d’écran de la sortie du terminal correspondante à l’exécution des commandes «git fetch» et «git log --graph --decorate --glob="\*" --oneline –name-only».





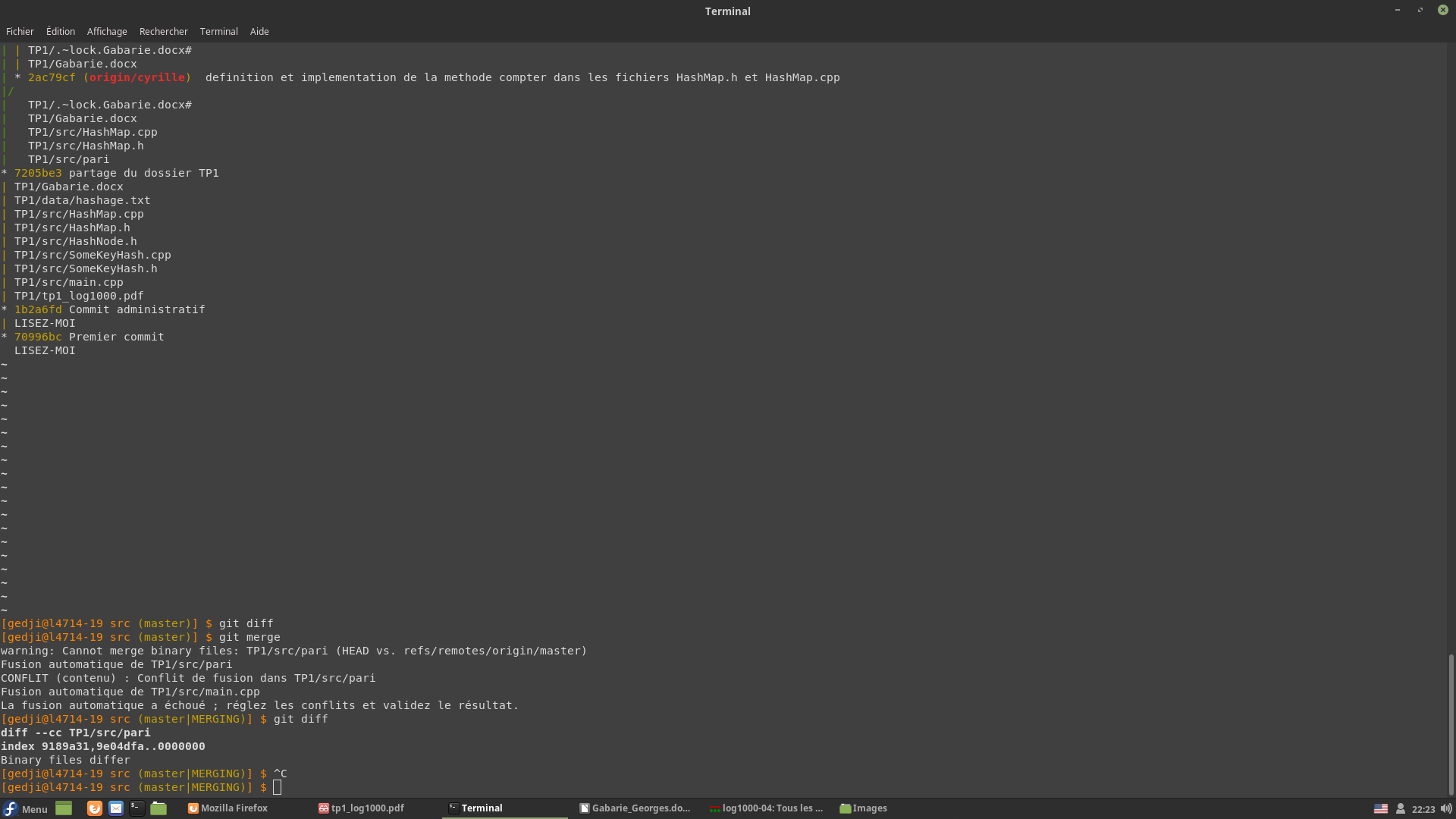
5-b) *Est-ce qu’il y aura un conflit lors d’un merge ? Pourquoi (pas) ? Si nécessaire, utilisez «*

*git diff » pour décider.*

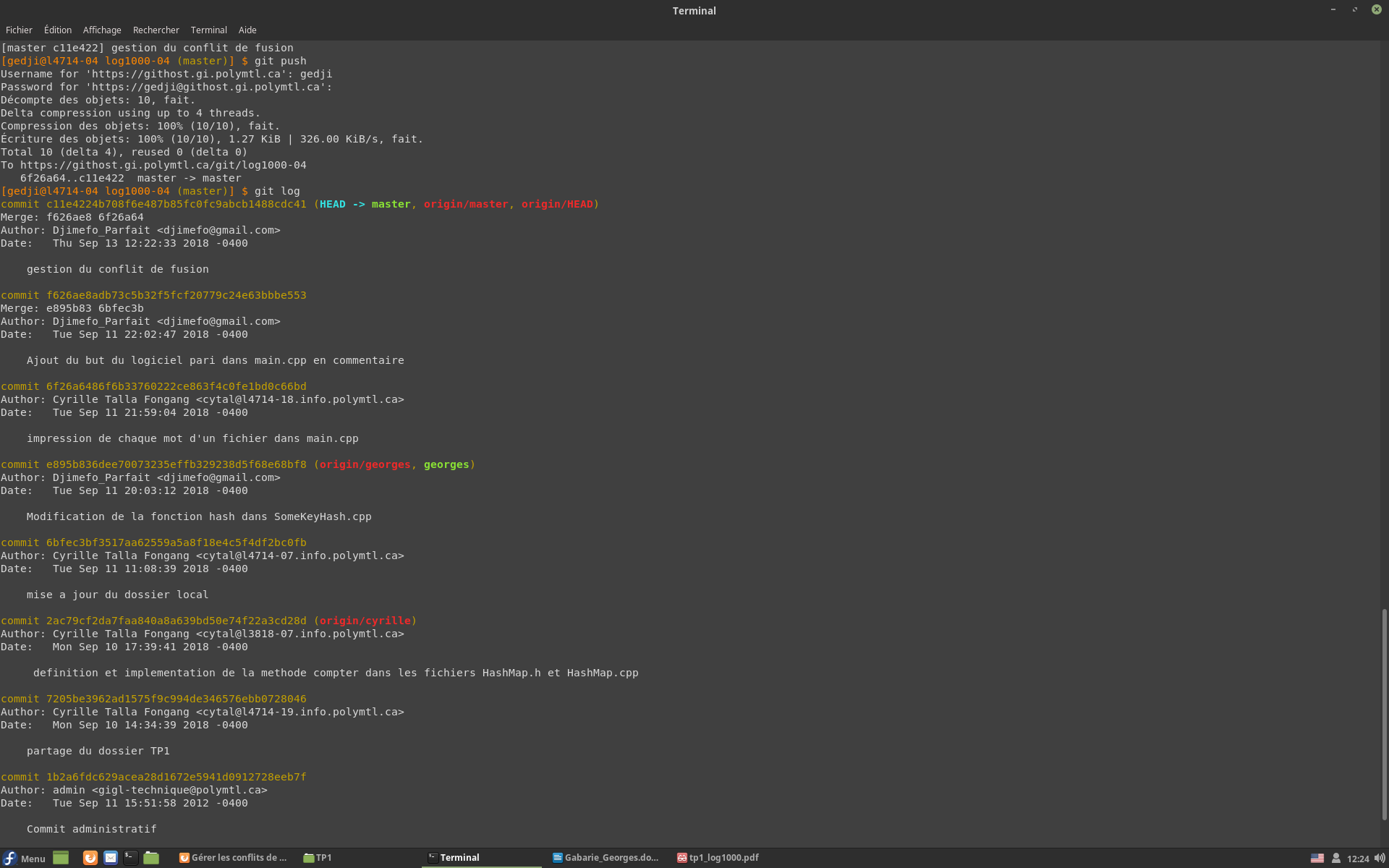
Oui il y aura un conflit du fait que la version de l’exécutable *«*pari*»* dans présent sur le serveur tient compte des modifications faites par l’équipier 1 (changement du comportement de la fonction main qui imprime maintenant chaque mot du fichier) tandis que celle qui va être poussée par l’équipier 2 n’inclut pas cette modification (puisque ce dernier n’a ajouté qu’un commentaire). Voir la capture d’écran a la question 5-c.

Il n’y aurait pas eu de conflit si l’on avait mis l’extension bin dans le fichier .gitignore; ce qui a été fait plus loin dans la suite du TP1.

*5-c) Exécution de la commande « git merge » et capture d'écran.*



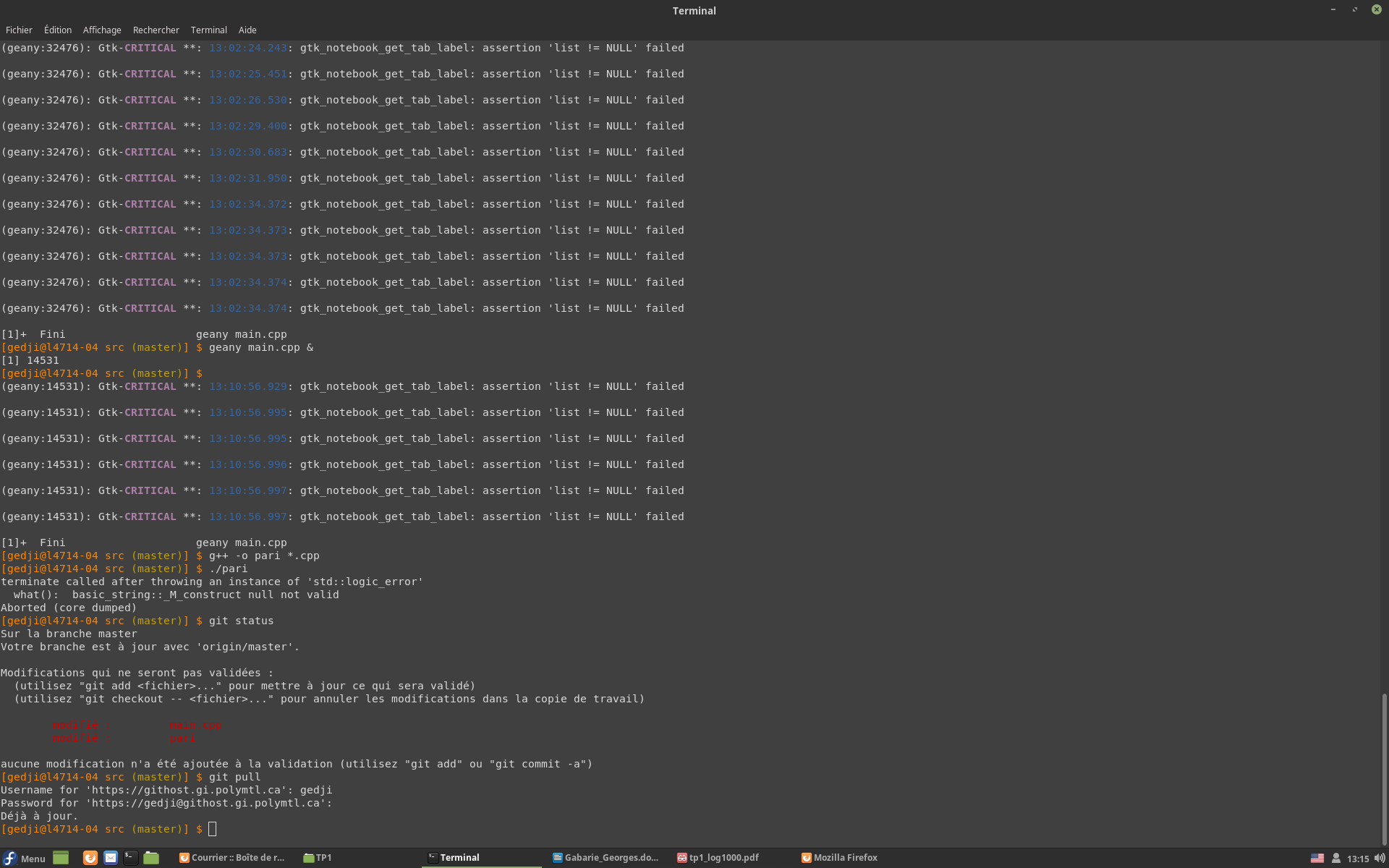
5-d) Exécution de la commande « git push » suivi de « git log » et une capture d’écran de la sortie



Il ne nous est pas possible de faire la comparaison du Log obtenu avec celui de la question E1.2 car dans le document de référence du TP (TP1\_LOG1000.pdf) ne figurait pas l’instruction de prendre une capture d’écran en E1.2.

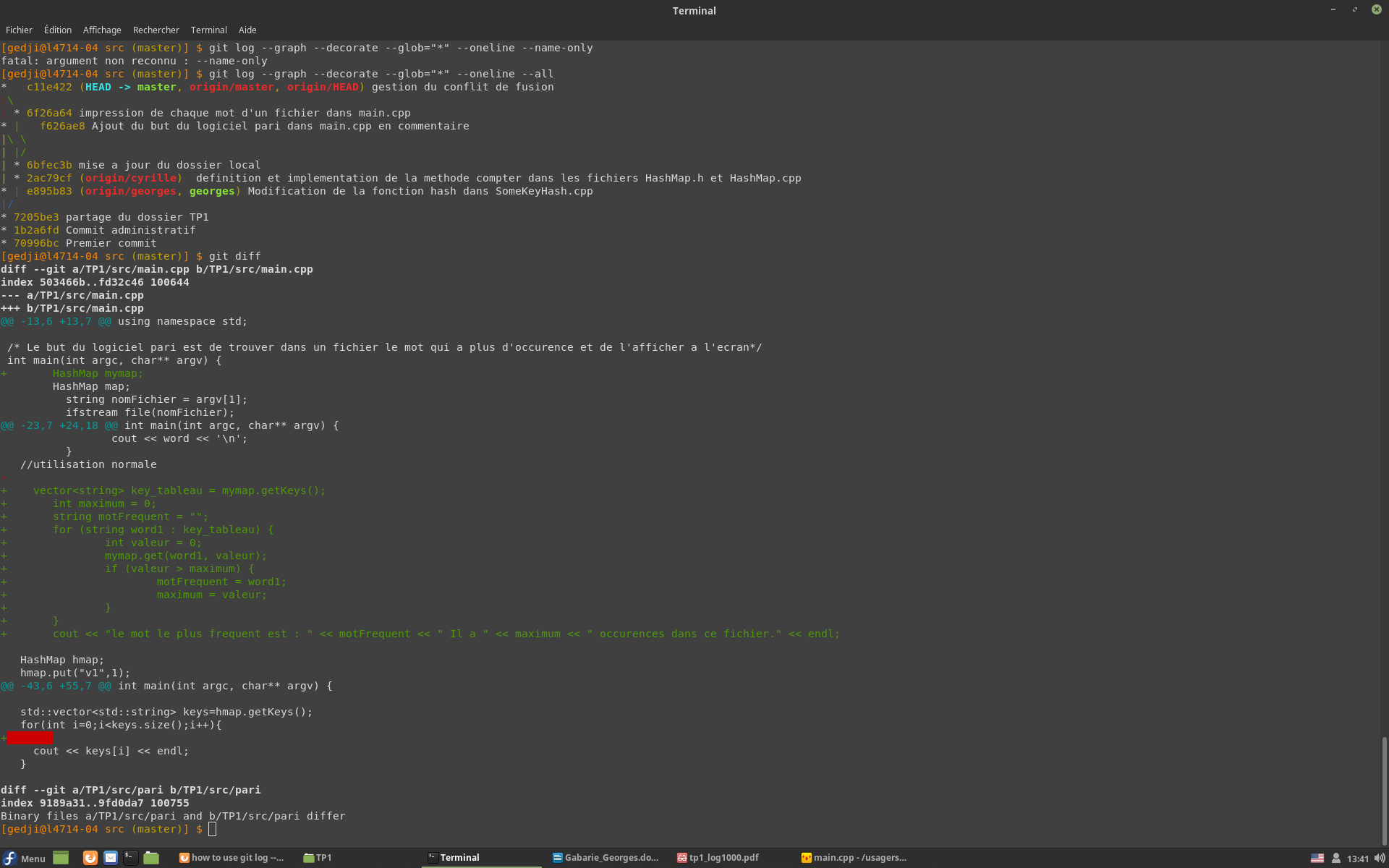
***E1.4 Modification 3***

*5-a) Capture d’écran de la sortie du terminal correspondante à l’exécution de la commande «git pull».*



*5-b) Est-ce que Git a détecté un conflit ? Pourquoi (pas)? Utilisez «git log --graph --decorate --glob="\*" --oneline --name-only​ » ​ et «git diff​ » ​ pour supporter votre explication.*

Non il n’a pas détecté de conflit car les modifications faites dans le répertoire local de l’équipier 2 n’ont pas été validées (dès lors qu’aucune commande git add, git commit n’a été exécutée depuis le dernier git pull).

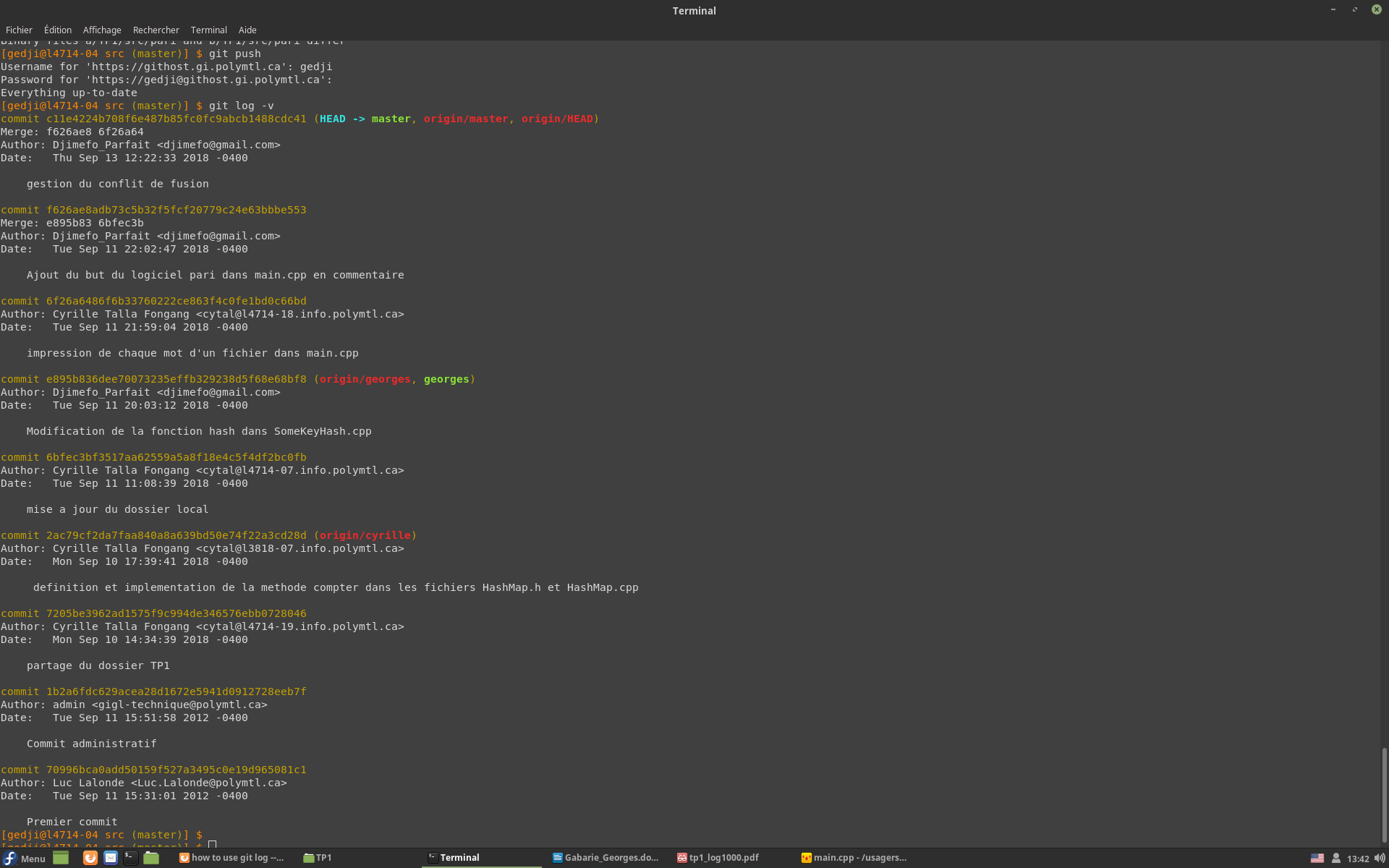


5-c) *Dans le cas où vous rencontrez une situation conflictuelle, comment pensez- vous régler cette dernière si on veut que le logiciel résultant réussit à résoudre le pari avec l’étudiant ? Est-ce que ça peut être fait automatiquement ? Pourquoi (pas) ? Si oui, résolvez le conflit.*

En cas de conflit, on ouvre le fichier faisant l’objet du conflit, on supprime les Heads>>>>>>>>>> qui ont été ajoutés par GIT, puis l’on décide quelle solution maintenir . Ensuite on fait git merge.

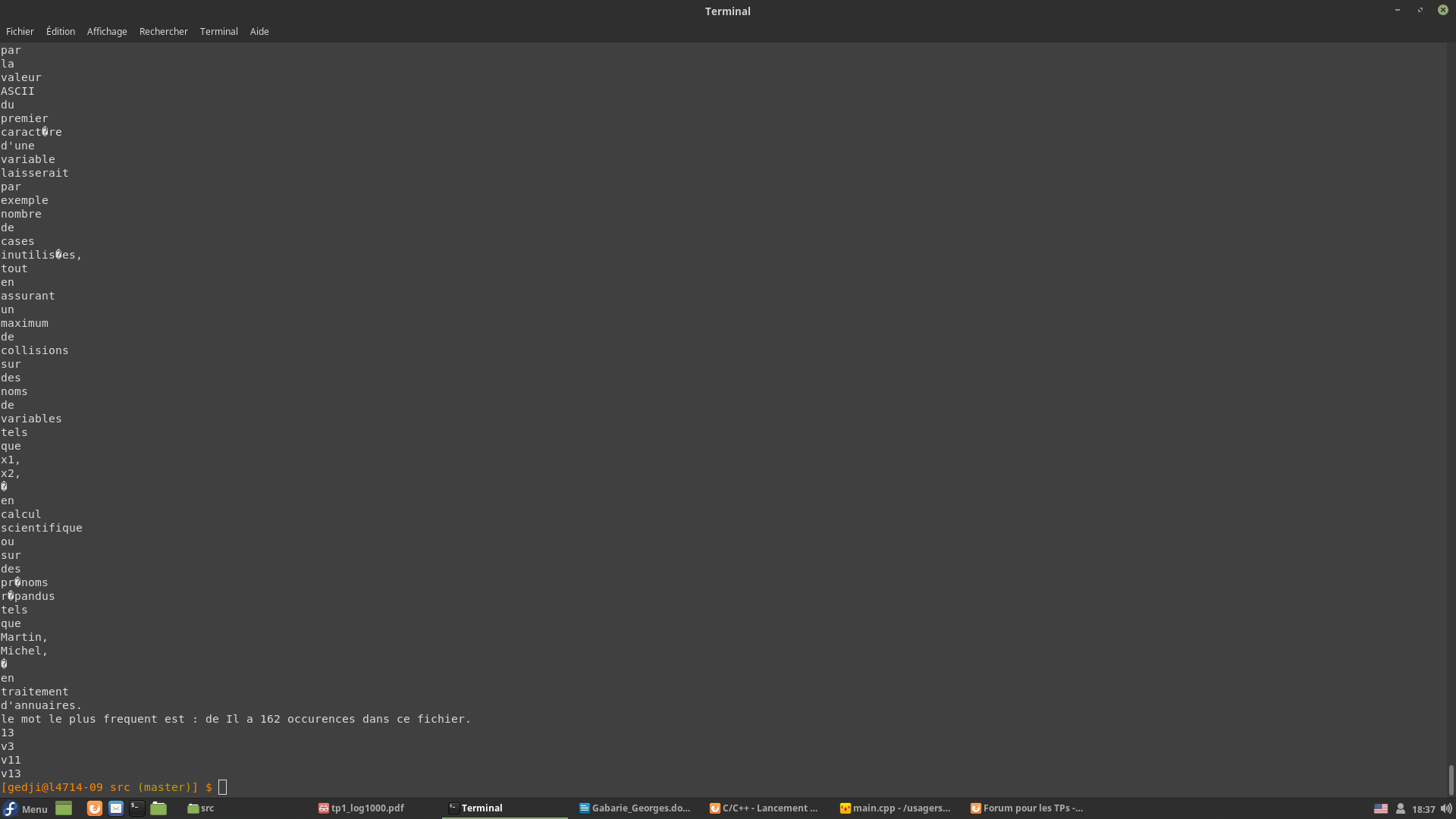
La résolution de ce conflit ne peut pas être fait automatiquement car Git ne décide pas *à*  notre place.

*6) Exécution de la commande « git push » suivi de « git log –v » et capture d’écran de la sortie correspondante.*



7) BONUS: Q*uel est le mot le plus populaire dans « The Art of Computer Programming»*

*Comme le montre la sortie d’écran ci dessous, le mot le plus populaire dans le fichier texte qui se trouve dans le dossier data est: «de» avec une fréquence de 162.*



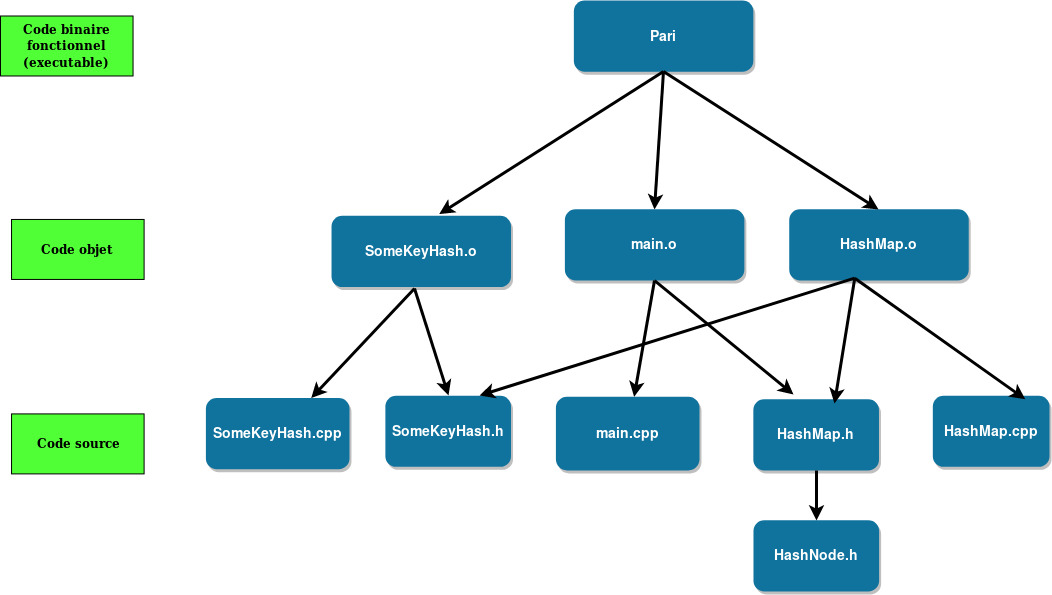
8) Description en 3 lignes max l'utilité des branches dans GIT.

Dans le développement d'un projet, il arrive souvent qu'on souhaite travailler sur une fonctionnalité en limitant les interactions avec le reste de l'équipe. Dans ce genre de situation les branches s‘avèrent utiles [2].

**E2. MAKE**

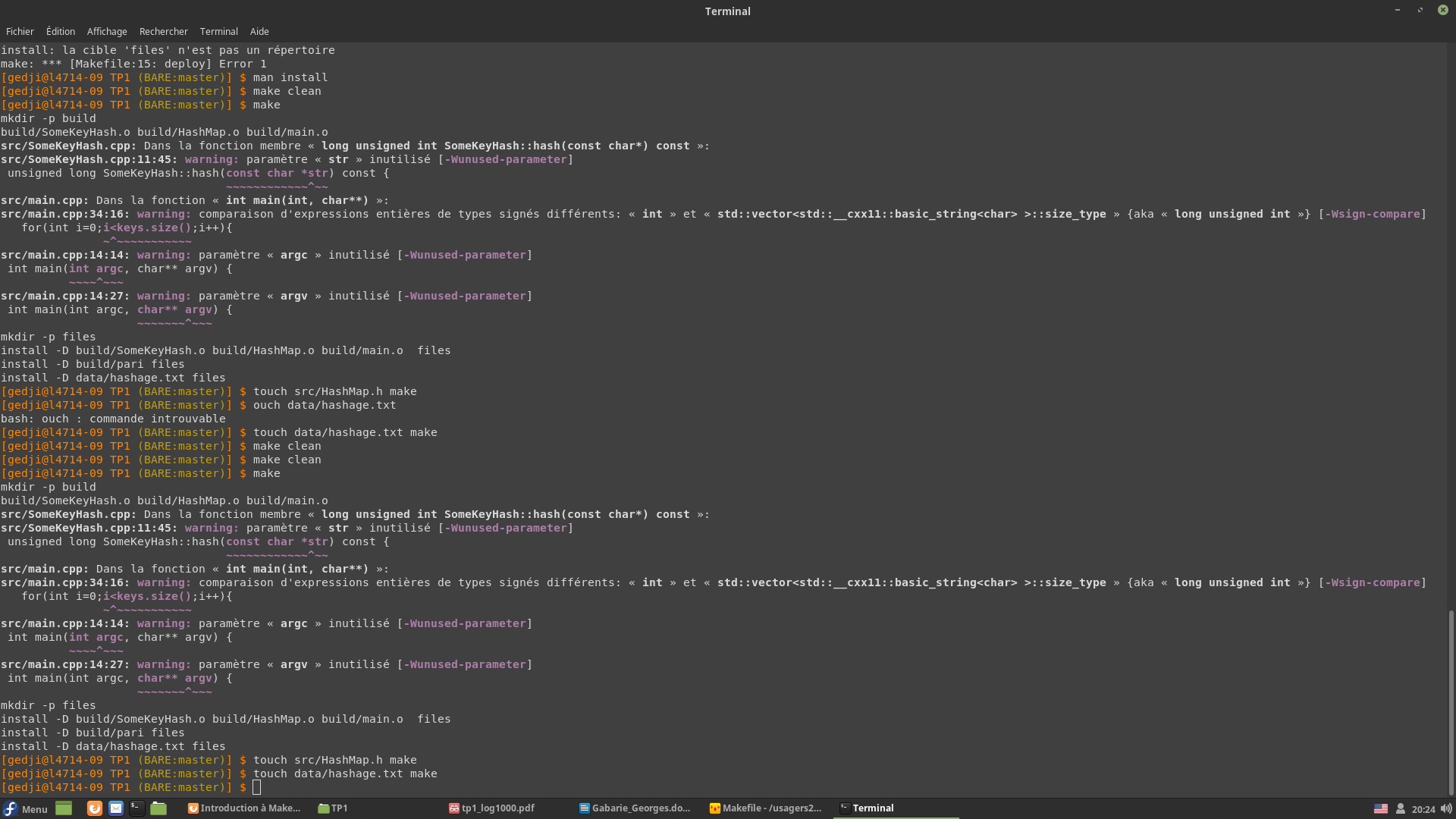
***E2.1 Éléments de construction d’un exécutable***

a) G*raphe de dépendances*



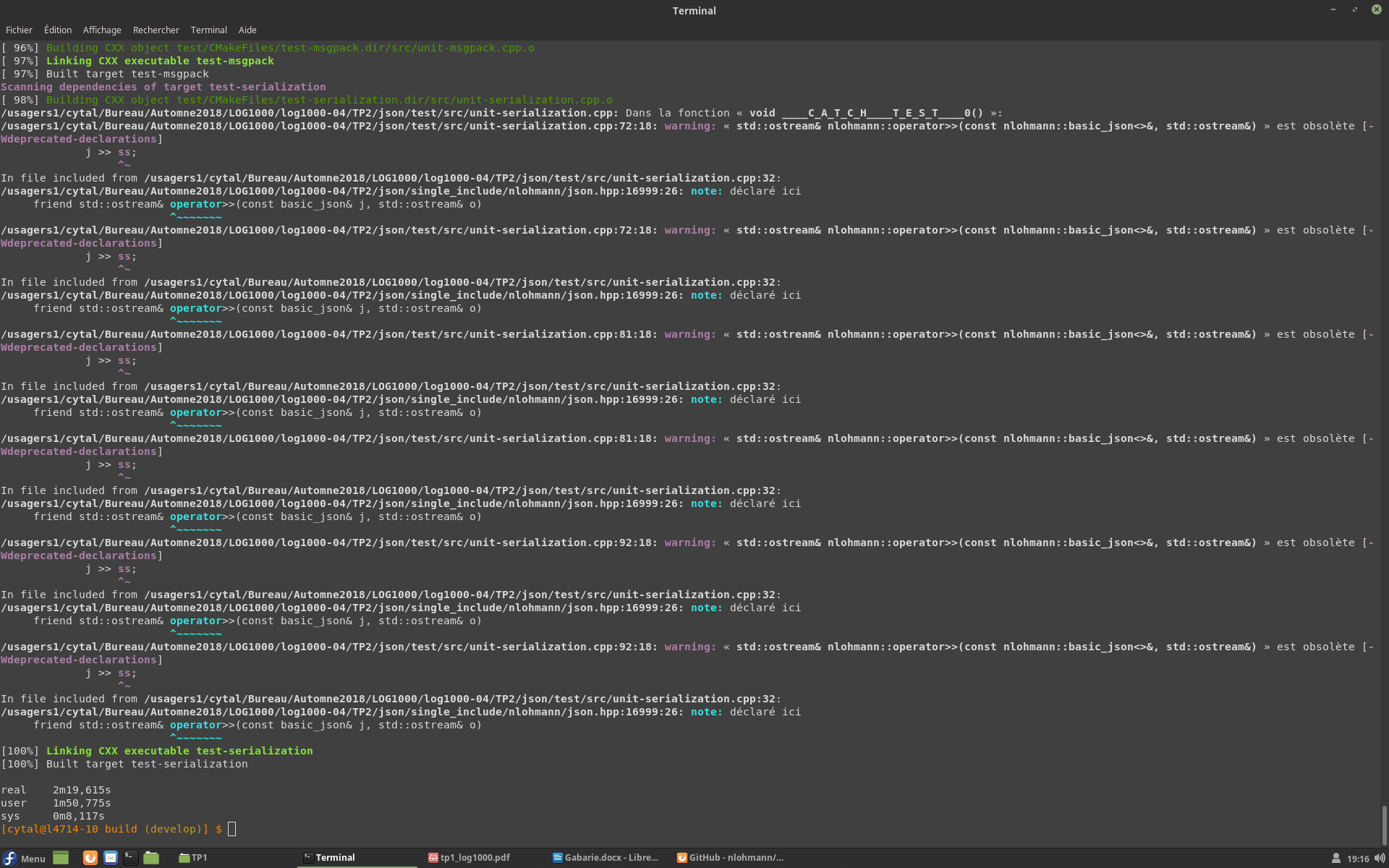
***E2.2 Création du Makefile et exécution du programme***

*4) Exécution de la commande* «make»*, et capture d’ecran des deux commandes* «touch … make»

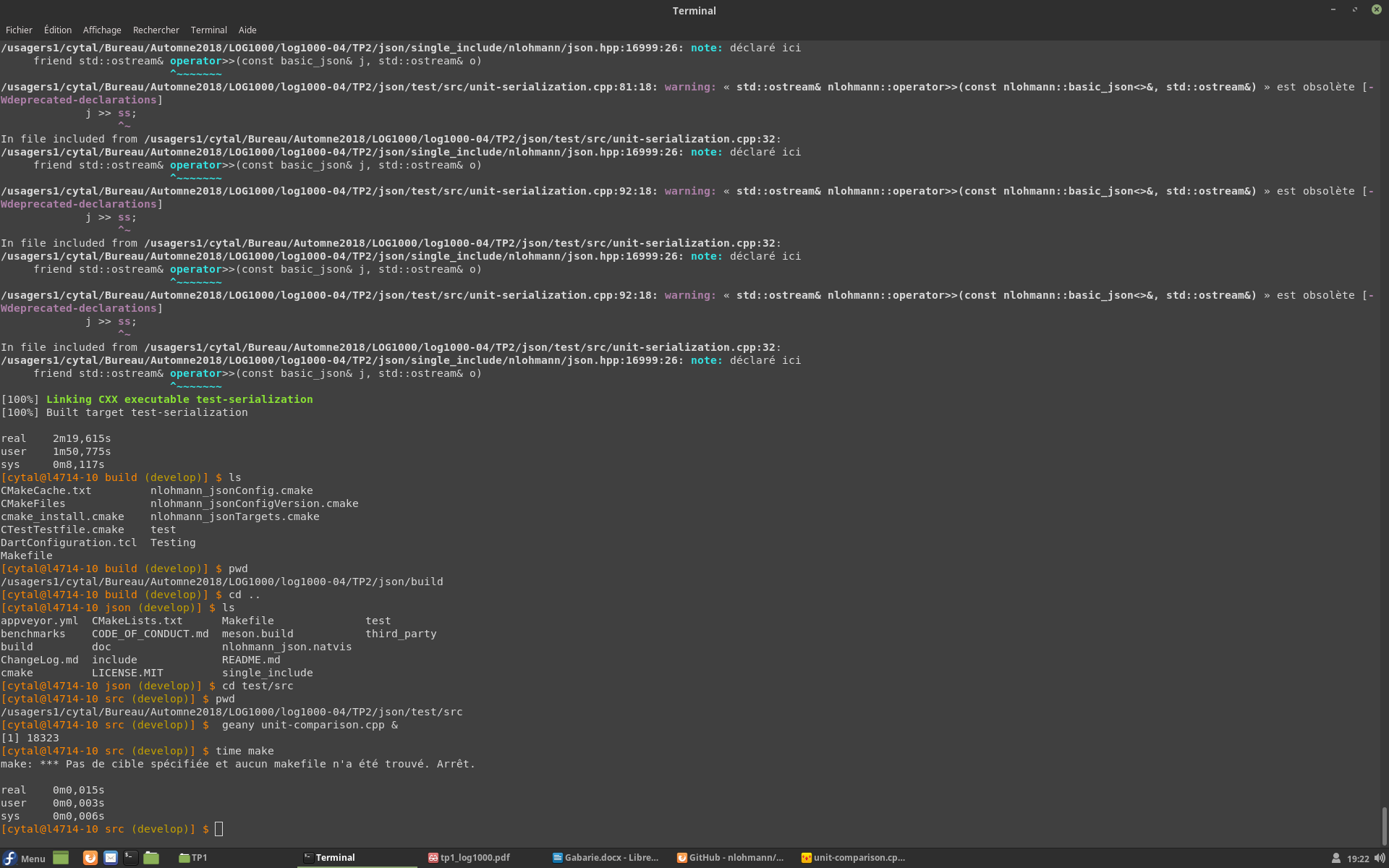


***Partie 2 :******E2.3 Un Makefile en JSON***

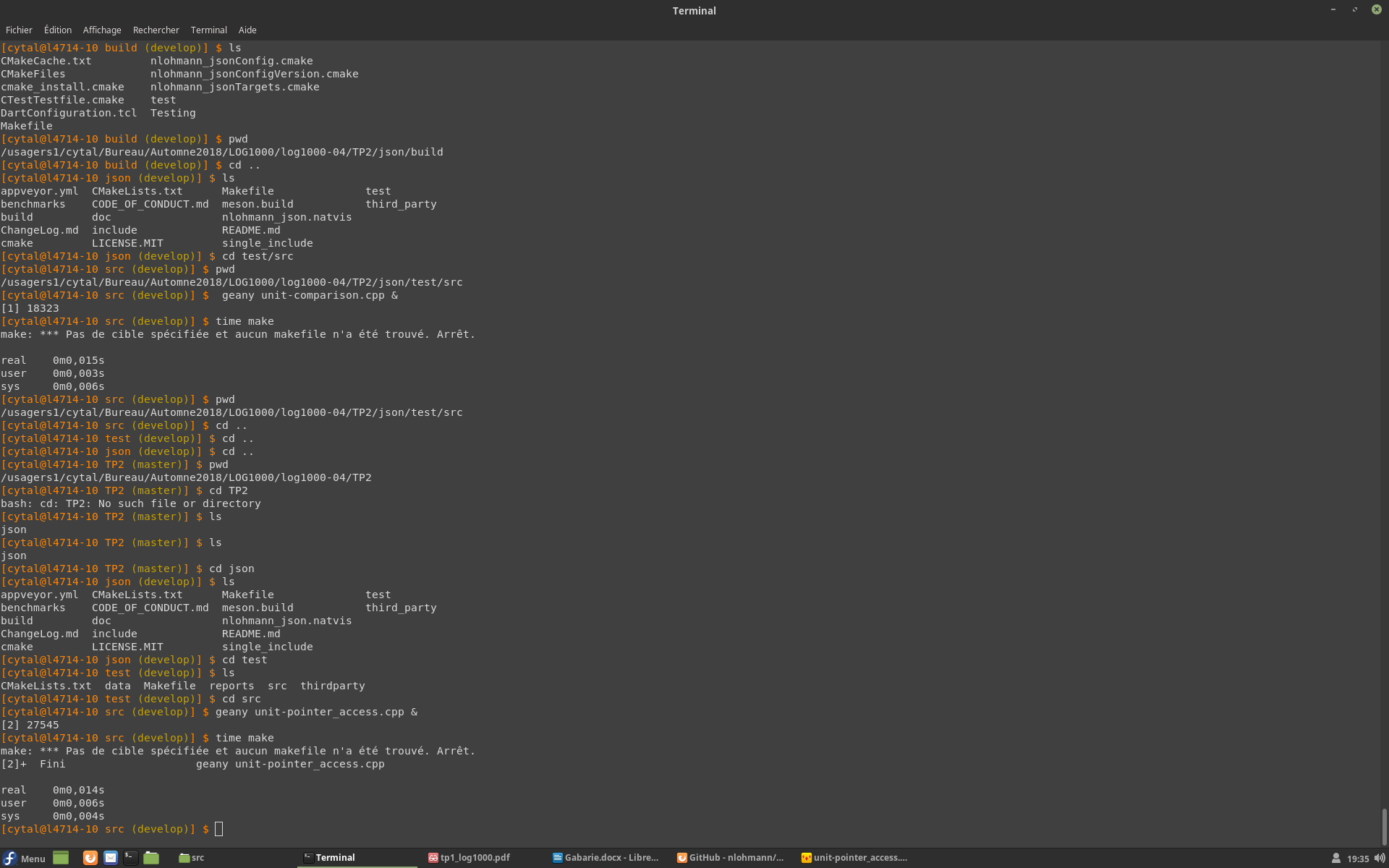
*2-a) Première exécution de la commande « time make » et capture de la sortie.*

**

*2-b) Seconde exécution de la commande « time make » et capture de la sortie.*

**

*2-c) Troisième exécution de la commande « time make » et capture de la sortie.*

**

*2-d)Est ce qu’il y a une différence de temps entre les trois exécutions de la commande ? Pourquoi (pas) ?*

Il y a une différence de temps entre les trois exécutions de la commande. En effet, la première commande a exécuté le programme avec plus de temps (2min19s) car il s’agissait de la première fois qu’il était compilé. Lors des autres commandes, il ne compile que les parties modifiées du programme et donc met moins de temps à le faire (15s et 14 s).

***Bibliographie:***

[1] https://openclassroom.com/fr/courses/1233741-gerez-vos-codes-sources-avec-git

[2] https://ensiwiki.ensimag.fr/index.php?title=G%C3%A9rer\_des\_branches\_avec\_Git

Pondération du travail pratique

Partie 1 :

Pour E1.1 à E1.4, la qualité de vos messages pour les «commits»: [ /3]

Pour E1.1 à E1.4, la qualité du code source: [ /2]

E1.1 [ /3]

a) [ /2]

b) [ /1]

E1.2 [ /10]

2-a) [ /3]

2-b) [ /1]

5-a) [ /3]

5-b) [ /2]

Copie du log: [ /1]

E1.3 [ /12]

3-a) [ /2]

3-b) [ /1]

3-c) [ /1]

5-a) [ /2]

5-b) [ /2]

5-c) [ /2]

5-d) [ /1]

Copie du log: [ /1]

E1.4 [ /10]

5-a) [ /2]

5-b) [ /2]

5-c) [ /2]

* + 1. BONUS [/1]
    2. [ /3]

E2.1 [ /5]

*Insérez votre graphe de dépendances ici* [ /5]

E2.2 [ /10]

Fichier Makefile [ /7]

*Exécutez* «make»*, puis insérez la sortie des deux commandes* «touch ... make» [ /3]

**Partie** 2 : [ /5]

1-a)  [ /2]

1-b)  [ /1]

1-c)  [ /1]

1-d)  [ /1]

\*vous devez insérer une **capture d'écran** de votre travail remis a l'aide de la commande «git *ls-files https://githost.gi.polymtl.ca/git/log1000-XX*»

**TOTAL: [ /60]**