**TRAVAIL PRATIQUE – MANIPULATION DE FICHIERS ET GESTION AVEC GIT**

**Nom de l'étudiant** : GOUIFE A DJIBEN ERIC CHRISTIAN Groupe : \_\_\_\_\_\_\_\_\_\_\_

**Consignes générales importantes :**

* **Chaque modification doit être suivie exactement des commandes Git suivantes dans l'ordre :**
  1. git branch nom\_branche
  2. git checkout nom\_branche
  3. git add chemin/fichier\_modifié
  4. git commit -m "Message clair expliquant la modification"
  5. Retour sur main : git checkout main
  6. Fusion de branche : git merge nom\_branche
  7. Suppression branche fusionnée : git branch -d nom\_branche
* Chaque commande Git utilisée doit être clairement inscrite dans git\_commands.txt.
* Tous les scripts Python doivent être dans le dossier scripts/.
* Tous les fichiers de données générés doivent être placés dans le dossier data/.

**1. Travailler avec des fichiers TEXTE (.txt)**

**1.1. INITIALISATION DU PROJET**

**Emplacement :** à la racine de votre espace de travail.

Complétez les commandes suivantes exactement :

➤ Vos commandes Git:

# Écrire ici vos commandes Git :

................................................................................

mkdir TP2

cd TP2

git init

git config –local user.name “Lenrics-01”

git config –local user.email “e2479537@cmaisonneuve.qc.ca”

➤ Exemple 1 :

Commande 1 : \_\_\_\_ # Créer un dossier nommé projet\_git\_text (nom explicite et sans espaces)

Commande 2 : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ # Se déplacer dans le dossier nouvellement créé

Commande 3 : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ # Initialiser un dépôt Git local dans ce dossier

Commande 4 : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ # Créer un fichier README.md contenant une courte description du projet

Commande 5 : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ # Ajouter le fichier README.md à la zone de staging (index Git)

Commande 6 : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ # Effectuer un commit initial avec un message clair (ex. : "Initialisation du projet avec README")

➤ Exemple 2 :

* Allez voir exemple.md dans le même dossier ici

**1.2 Écriture sans utiliser with open()**

**Branche Git à créer :** txt-sans-with

**Emplacement :**

* Script : scripts/ecriture\_sans\_with.py
* Fichier : data/journal.txt
* Écrivez dans le fichier (journal.txt) sans utiliser with open() :

Jour 1 : Introduction

Jour 2 : Variables

Jour 3 : Conditions

Jour 4 : Boucles

Jour 5 : Fonctions

Jour 6 : Listes

Jour 7 : Dictionnaires

Jour 8 : Tuples

Jour 9 : Ensembles

Jour 10 : Manipulation de fichiers texte

Jour 11 : Manipulation de fichiers CSV

Jour 12 : Manipulation de fichiers JSON

Jour 13 : NumPy - Sauvegarde de tableaux

Jour 14 : NumPy - Chargement de tableaux

Jour 15 : Graphiques avec matplotlib

Jour 16 : Résolution d'erreurs JSON

Jour 17 : Gestion des exceptions en Python

Jour 18 : Écriture avancée avec format()

Jour 19 : Encodage de fichiers texte

Jour 20 : Conclusion et révision

➤ Votre script python :

# Écrire ici le script Python :

................................................................................

*Commandes Git à effectuer :* (branch, checkout, add, commit, retour sur main, merge, branch -d)

➤ Vos commandes Git:

# Écrire ici vos commandes Git :

................................................................................

**1.3 Réécriture avec utilisation de with open()**

**Branche Git à créer :** txt-avec-with

**Emplacement :**

* Script : scripts/ecriture\_avec\_with.py
* Fichier modifié : data/journal.txt
* Réécrivez les mêmes lignes précédentes avec with open().
* Ajoutez ensuite une ligne en mode "a" : Jour 21 : Examen 1

➤ *Votre script python et vos Commandes Git à effectuer :* (branch, checkout, add, commit, retour sur main, merge, branch -d)

# Écrire ici le script Python et les commandes Git dans l'ordre :

................................................................................

**1.4 Lecture numérotée du fichier texte**

**Branche Git à créer :** txt-lecture

**Emplacement :**

* Script : scripts/lecture\_journal.py
* Affichez chaque ligne de journal.txt avec le numéro de ligne :

Ligne 1 : Jour 1 : Introduction

Ligne 2 : Jour 2 : Variables

Ligne 3 : Jour 3 : Conditions

Ligne 4 : Jour 4 : Boucles

....

➤ *Votre script python et vos Commandes Git à effectuer :* (branch, checkout, add, commit, retour sur main, merge, branch -d)

# Écrire ici le script Python et les commandes Git dans l'ordre :

................................................................................

**2. FICHIERS CSV (.csv)**

**2.1 Création CSV avec csv.writer()**

**Branche Git à créer :** csv-creation

**Emplacement :**

* Script : scripts/creation\_csv.py
* Fichier généré : data/notes.csv

Créez un fichier CSV contenant exactement :

Nom,Note

Alice,90

Bob,72

Chloe,88

...

➤ *Votre script python et vos Commandes Git à effectuer :* (branch, checkout, add, commit, retour sur main, merge, branch -d)

# Écrire ici le script Python et les commandes Git dans l'ordre :

................................................................................

**2.2 Lecture brute du CSV (texte brut)**

**Branche Git à créer :** csv-brut

**Emplacement :**

* Script : scripts/lecture\_brute.py
* Lisez et affichez simplement le contenu du fichier CSV comme texte brut avec open().

➤ *Votre script python et vos Commandes Git à effectuer :* (branch, checkout, add, commit, retour sur main, merge, branch -d)

# Écrire ici le script Python et les commandes Git dans l'ordre :

................................................................................

**2.3 Lecture structurée CSV (csv.reader())**

**Branche Git à créer :** csv-reader

**Emplacement :**

* Script : scripts/lecture\_structurée.py
* Lisez le CSV et affichez clairement sous la forme suivante :

Alice a obtenu 90

Bob a obtenu 72

Chloe a obtenu 88

➤ *Votre script python et vos Commandes Git à effectuer :* (branch, checkout, add, commit, retour sur main, merge, branch -d)

# Écrire ici le script Python et les commandes Git dans l'ordre :

................................................................................

**2.4 Conflit Git volontaire à créer**

* Branche ajout-etudiant-1 (créée depuis main)  
  Ajoutez David,95 dans notes.csv puis committez.
* Branche ajout-etudiant-2 (créée depuis main)  
  Ajoutez Emma,91 dans notes.csv puis committez.
* Fusionnez ces deux branches dans main pour provoquer un conflit.  
  Résolvez le conflit manuellement et expliquez votre résolution dans analyse.txt.

**Commandes Git à effectuer :** branch, checkout, add, commit, checkout main, merge, résoudre conflit, branch -d

➤ Votre réponse

# Écrire ici le script Python et les commandes Git dans l'ordre :

................................................................................

**3. FICHIERS JSON (.json)**

**3.1 Création JSON**

**Branche Git à créer :** json-creation

**Emplacement :**

* Fichier JSON créé : data/produits.json

Créez exactement :

{

"produits": [

{"nom": "Clé USB", "prix": 12.5},

{"nom": "Stylo", "prix": 2.0}

]

}

**Commandes Git à effectuer** (complètes)

➤ Vos Commandes Git à effectuer :

# Écrire ici le script Python et les commandes Git dans l'ordre :

................................................................................

**3.2 Conflit Git volontaire JSON à créer**

* Branche json-promo (depuis main) : Ajoutez "promo": true à "Clé USB". Committez.
* Branche json-prix (depuis main) : Changez "prix" de "Clé USB" à 10.0. Committez.
* Fusionnez ces deux branches dans main.
* Résolvez le conflit manuellement en conservant les deux changements.
* Documentez clairement votre résolution dans analyse.txt.

**4. FICHIERS NUMPY (.csv avec numpy)**

**4.1 Sauvegarde tableau NumPy**

**Branche Git à créer :** numpy-save

**Emplacement :**

* Script : scripts/save\_numpy.py
* Fichier généré : data/fibonacci.csv
* Sauvegardez le tableau [1, 2, 3, 5, 8] avec np.savetxt().

**4.2 Chargement et analyse tableau NumPy**

**Branche Git à créer :** numpy-load-analyse

**Emplacement :**

* Script : scripts/load\_analyse\_numpy.py
* Chargez fibonacci.csv, puis affichez la moyenne, le minimum, le maximum.

➤ *Votre script python et vos Commandes Git à effectuer :* (branch, checkout, add, commit, retour sur main, merge, branch -d)

# Écrire ici le script Python et les commandes Git dans l'ordre :

................................................................................

**5. QUESTIONS FINALES (analyse.txt à compléter précisément)**

1. Pourquoi est-il conseillé d'utiliser with open() plutôt que open()/close() ? Réponse : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
2. Expliquez précisément comment vous avez résolu les conflits Git créés dans ce TP : Réponse : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
3. Donnez la commande Git pour restaurer un fichier supprimé accidentellement : Réponse : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**6. VÉRIFICATION FINALE**

Exécutez cette commande exactement dans votre dépôt à la fin du TP :

git log --oneline --graph

Copiez le résultat exact dans l'espace suivant :

# Collez ici le résultat exact de git log --oneline --graph :

**7. QUESTION FINALE – SCÉNARIO RÉALISTE AVEC DONNÉES COMPLEXES**

**7.1. Objectif pédagogique**

Mobiliser vos compétences sur :

* Lecture et filtrage de données CSV avec en-tête
* Traitement de données numériques avec NumPy
* Commandes Git avancées : suppression, restauration, historique, résolution de conflit
* Automatisation d’une analyse avec génération conditionnelle d’un fichier JSON

**7.2. CONTEXTE DU SCÉNARIO :**

Vous travaillez sur un fichier de résultats data/reponses.csv, contenant les réponses à un questionnaire de satisfaction. Il est structuré comme suit :

Nom,Prénom,Note

Garcia,Ana,4

Dubois,Jean,5

Nguyen,Linh,4

Smith,John,3

...

* Le fichier contient **20 lignes de réponses valides**.
* Le fichier complet peut être copié de l'annexe ci-bas (fichier responses.csv)
* Il doit être **commité dans main** au début du projet.

Un autre fichier data/meta.json contient les métadonnées associées :

{

"questionnaire": "Évaluation du cours",

"type": "satisfaction",

"nb\_repondants": 20

}

**7.3. ÉTAPE 1 – Suppression accidentelle d’un fichier critique**

1. À partir de main, créez la branche analyse-1.
2. Dans cette branche, **supprimez** le fichier data/reponses.csv :

rm data/reponses.csv

git rm data/reponses.csv

git commit -m "Suppression accidentelle du fichier de réponses"

**7.4. ÉTAPE 2 – Échec d’analyse dans une autre branche**

1. Revenez dans main, puis créez la branche analyse-2.
2. Dans cette branche, vous tentez de faire une analyse avec ce script :

import numpy as np

data = np.loadtxt("data/reponses.csv", delimiter=",", skiprows=1, usecols=2)

Cela génère une **erreur**, car le fichier a été supprimé dans l’autre branche.

**7.5. ÉTAPE 3 – Restauration correcte avec Git**

1. **Identifiez le commit** dans lequel le fichier existait encore (git log, git checkout, etc.).
2. **Restaurez** le fichier supprimé dans la branche analyse-2, sans le recréer manuellement :

git checkout main data/reponses.csv

1. Commitez cette récupération :

git add data/reponses.csv

git commit -m "Restauration du fichier supprimé depuis l’historique"

**7.6. ÉTAPE 4 – Script d’analyse avancée**

Créez le fichier suivant :  
scripts/analyse\_satisfaction.py

Il doit :

* Lire la colonne Note du fichier CSV (usecols=2)
* Calculer :
  + Moyenne des notes
  + Médiane
  + Pourcentage de notes supérieures ou égales à 4
* Afficher les résultats sous forme lisible
* Si la moyenne est inférieure à 3.0, **générer automatiquement un fichier data/alerte.json** :

{

"alerte": true,

"moyenne": 2.6,

"seuil": 3.0,

"message": "Satisfaction étudiante anormalement basse"

}

Sinon, **ne rien créer**.

**7.7. À REMPLIR DANS VOTRE RAPPORT**

1. Script complet scripts/analyse\_satisfaction.py :

# Script Python :

................................................................................

1. Liste complète des commandes Git utilisées (suppression, checkout, restauration, commit, etc.) :

# Commandes Git :

................................................................................

1. Contenu final du fichier data/reponses.csv (après restauration) :

# Contenu CSV :

................................................................................

1. Si généré, collez le contenu de data/alerte.json :

# JSON :

................................................................................

**Rappels importants :**

* Ne modifiez jamais manuellement les fichiers supprimés : utilisez Git.
* np.loadtxt() ou np.genfromtxt() peuvent tous deux être utilisés.
* N’oubliez pas skip\_header=1 si vous avez une ligne d’en-tête.
* Si vous utilisez csv.reader, veillez à convertir les notes en int ou float.

**8. LIVRABLES À FOURNIR EXACTEMENT**

**RAPPORT À REMETTRE**

Remettez **un seul fichier** nommé :

tp\_fichiers\_NOM\_PRENOM.docx (ou .pdf, .txt ou .py)

Le rapport doit contenir :

* Le **code Python complet** pour chaque étape (copié dans le document)
* Les **commandes Git** associées à chaque script (branch, checkout, add, commit, merge, branch -d)
* Les **résolutions de conflits Git**, avec explication claire
* Les **réponses aux questions finales**
* Le **résultat complet de** :
* git log --oneline --graph

Aucun .zip ou dossier séparé n’est accepté. Le code et les commandes doivent être intégrés dans ce seul fichier.

**Vérifiez soigneusement chaque étape avant remise du TP.**

**Annexe : Fichiers à utiliser dans le TP**

**data/journal.txt**

Jour 1 : Introduction

Jour 2 : Variables

Jour 3 : Conditions

Jour 4 : Boucles

Jour 5 : Fonctions

Jour 6 : Listes

Jour 7 : Dictionnaires

Jour 8 : Tuples

Jour 9 : Ensembles

Jour 10 : Manipulation de fichiers texte

Jour 11 : Manipulation de fichiers CSV

Jour 12 : Manipulation de fichiers JSON

Jour 13 : NumPy - Sauvegarde de tableaux

Jour 14 : NumPy - Chargement de tableaux

Jour 15 : Graphiques avec matplotlib

Jour 16 : Résolution d'erreurs JSON

Jour 17 : Gestion des exceptions en Python

Jour 18 : Écriture avancée avec format()

Jour 19 : Encodage de fichiers texte

Jour 20 : Conclusion et révision

**data/notes.csv**

Nom,Note

Alice,90

Bob,72

Chloe,88

**data/produits.json**

{

"produits": [

{"nom": "Clé USB", "prix": 12.5},

{"nom": "Stylo", "prix": 2.0}

]

}

**data/responses.csv**

Nom,Prénom,Note

Garcia,Ana,4

Dubois,Jean,5

Nguyen,Linh,4

Smith,John,3

Tremblay,Marie,2

Zhao,Yan,1

Roy,Éric,4

Mendes,Carlos,5

Bouchard,Laurie,5

Khan,Imran,3

Ouellet,Lucie,4

Kim,Min,2

Martin,Chloé,1

Benoit,Paul,2

Bergeron,Julie,3

Côté,Alex,5

Lavoie,Noémie,5

Ahmed,Yasmin,4

Gagnon,Sylvain,3

Lemay,Maude,2