

# Module Programmation et Acquisition

Clément **Bâty**

11 décembre 2023

- 1 Planning [Rappels]
- 2 GIT : système de gestion de version
- 3 Démarrage du projet

# Plan

- 1 Planning [Rappels]
- 2 GIT : système de gestion de version
  - Présentation rapide
  - HowTo GIT (Local)
  - Mini TP GIT
  - Autre manière de travailler avec GIT
- 3 Démarrage du projet
  - Projet
  - Évaluation

# Calendrier détaillé

## Rétroplanning !

- 6 novembre : bases de python + intro

# Calendrier détaillé

## Rétroplanning !

- 6 novembre : bases de python + intro
- 13 novembre : premiers pas en POO et PyQt + DM

# Calendrier détaillé

## Rétroplanning !

- 6 novembre : bases de python + intro
- 13 novembre : premiers pas en POO et PyQt + DM
- 4 décembre : DM corrigé + convention, + doc + modules sciences

# Calendrier détaillé

## Rétroplanning !

- 6 novembre : bases de python + intro
- 13 novembre : premiers pas en POO et PyQT + DM
- 4 décembre : DM corrigé + convention, + doc + modules sciences
- 11 décembre : GIT, attentes et démarrage du projet I/IV

# Calendrier détaillé

## Rétroplanning !

- 6 novembre : bases de python + intro
- 13 novembre : premiers pas en POO et PyQT + DM
- 4 décembre : DM corrigé + convention, + doc + modules sciences
- 11 décembre : GIT, attentes et démarrage du projet I/IV
- 22 janvier : séance de travail II/IV



# Calendrier détaillé

## Rétroplanning !

- 6 novembre : bases de python + intro
- 13 novembre : premiers pas en POO et PyQT + DM
- 4 décembre : DM corrigé + convention, + doc + modules sciences
- 11 décembre : GIT, attentes et démarrage du projet I/IV
- 22 janvier : séance de travail II/IV
- 29 janvier : séance de travail [2H30] III/IV

# Calendrier détaillé

## Rétroplanning !

- 6 novembre : bases de python + intro
- 13 novembre : premiers pas en POO et PyQT + DM
- 4 décembre : DM corrigé + convention, + doc + modules sciences
- 11 décembre : GIT, attentes et démarrage du projet I/IV
- 22 janvier : séance de travail II/IV
- 29 janvier : séance de travail [2H30] III/IV
- 12 mars : séance de finition [2H30] IV/IV

# Calendrier détaillé

## Rétroplanning !

- 6 novembre : bases de python + intro
- 13 novembre : premiers pas en POO et PyQT + DM
- 4 décembre : DM corrigé + convention, + doc + modules sciences
- 11 décembre : GIT, attentes et démarrage du projet I/IV
- 22 janvier : séance de travail II/IV
- 29 janvier : séance de travail [2H30] III/IV
- 12 mars : séance de finition [2H30] IV/IV

# Calendrier détaillé

## Rétroplanning !

- 6 novembre : bases de python + intro
- 13 novembre : premiers pas en POO et PyQt + DM
- 4 décembre : DM corrigé + convention, + doc + modules sciences
- 11 décembre : GIT, attentes et démarrage du projet I/IV
- 22 janvier : séance de travail II/IV
- 29 janvier : séance de travail [2H30] III/IV
- 12 mars : séance de finition [2H30] IV/IV

→ Date à prévoir pour le rendu (hors des séances) probablement  
**la semaine du 13 mai**

# Calendrier détaillé

## Rétroplanning !

- 6 novembre : bases de python + intro
- 13 novembre : premiers pas en POO et PyQt + DM
- 4 décembre : DM corrigé + convention, + doc + modules sciences
- 11 décembre : GIT, attentes et démarrage du projet I/IV
- 22 janvier : séance de travail II/IV
- 29 janvier : séance de travail [2H30] III/IV
- 12 mars : séance de finition [2H30] IV/IV

→ Date à prévoir pour le rendu (hors des séances) probablement  
**la semaine du 13 mai**

- Questionnaire en ligne (sur Moodle ?) : bilan (probablement en mars)

# Plan

- 1 Planning [Rappels]
- 2 GIT : système de gestion de version
  - Présentation rapide
  - HowTo GIT (Local)
  - Mini TP GIT
  - Autre manière de travailler avec GIT
- 3 Démarrage du projet
  - Projet
  - Évaluation

# Plan

- 1 Planning [Rappels]
- 2 GIT : système de gestion de version
  - Présentation rapide
  - HowTo GIT (Local)
  - Mini TP GIT
  - Autre manière de travailler avec GIT
- 3 Démarrage du projet
  - Projet
  - Évaluation

# GIT c'est quoi

I/III

... un gestionnaire de version

## GIT (d'après Wikipedia)

Git est un logiciel de gestion de versions décentralisé. C'est un logiciel libre créé par Linus Torvalds, auteur du noyau Linux, et distribué sous GPL V2.



# GIT c'est quoi

I/III

... un gestionnaire de version

## GIT (d'après Wikipedia)

Git est un logiciel de gestion de versions décentralisé. C'est un logiciel libre créé par Linus Torvalds, auteur du noyau Linux, et distribué sous GPL V2.

## C'est à dire ?

Un logiciel de gestion de versions<sup>a</sup> est :

- un logiciel qui permet de stocker un ensemble de fichiers

# GIT c'est quoi

I/III

... un gestionnaire de version

## GIT (d'après Wikipedia)

Git est un logiciel de gestion de versions décentralisé. C'est un logiciel libre créé par Linus Torvalds, auteur du noyau Linux, et distribué sous GPL V2.

## C'est à dire ?

Un logiciel de gestion de versions<sup>a</sup> est :

- un logiciel qui permet de stocker un ensemble de fichiers
- en conservant la chronologie de toutes les modifications qui ont été effectuées dessus.

# GIT c'est quoi

I/III

... un gestionnaire de version

## GIT (d'après Wikipedia)

Git est un logiciel de gestion de versions décentralisé. C'est un logiciel libre créé par Linus Torvalds, auteur du noyau Linux, et distribué sous GPL V2.

## C'est à dire ?

Un logiciel de gestion de versions<sup>a</sup> est :

- un logiciel qui permet de stocker un ensemble de fichiers
- en conservant la chronologie de toutes les modifications qui ont été effectuées dessus.

# GIT c'est quoi

I/III

... un gestionnaire de version

## GIT (d'après Wikipedia)

Git est un logiciel de gestion de versions décentralisé. C'est un logiciel libre créé par Linus Torvalds, auteur du noyau Linux, et distribué sous GPL V2.

## C'est à dire ?

Un logiciel de gestion de versions<sup>a</sup> est :

- un logiciel qui permet de stocker un ensemble de fichiers
- en conservant la chronologie de toutes les modifications qui ont été effectuées dessus.

→ logiciel enregistrant les  $\neq$  versions des modifications des fichiers !

---

a. (ou VCS en anglais, pour version control system)

# GIT c'est quoi

II/III

... utilisé par BEAUCOUP de monde

# GIT c'est quoi

II/III

... utilisé par BEAUCOUP de monde

Et quand je dis beaucoup ...

En 2016, il s'agit du logiciel de gestion de versions le plus populaire qui est utilisé par **plus de douze millions** de personnes dans le monde

# GIT c'est quoi

II/III

... utilisé par BEAUCOUP de monde

Et quand je dis beaucoup ...

En 2016, il s'agit du logiciel de gestion de versions le plus populaire qui est utilisé par **plus de douze millions** de personnes dans le monde

c'est pour des projets petits à énormes

- De la sauvegarde des modifs d'UN fichier

# GIT c'est quoi

II/III

... utilisé par BEAUCOUP de monde

Et quand je dis beaucoup ...

En 2016, il s'agit du logiciel de gestion de versions le plus populaire qui est utilisé par **plus de douze millions** de personnes dans le monde

c'est pour des projets petits à énormes

- De la sauvegarde des modifs d'UN fichier
- ou de toute la configuration de votre ordinateur



# GIT c'est quoi

II/III

... utilisé par BEAUCOUP de monde

Et quand je dis beaucoup ...

En 2016, il s'agit du logiciel de gestion de versions le plus populaire qui est utilisé par **plus de douze millions** de personnes dans le monde

c'est pour des projets petits à énormes

- De la sauvegarde des modifs d'UN fichier
- ou de toute la configuration de votre ordinateur
- à la création du noyau linux (milliers de devs, dizaines de milliers de patches, ...)

# GIT c'est quoi



... utilisé par BEAUCOUP de monde

Et quand je dis beaucoup ...

En 2016, il s'agit du logiciel de gestion de versions le plus populaire qui est utilisé par **plus de douze millions** de personnes dans le monde

c'est pour des projets petits à énormes

- De la sauvegarde des modifs d'UN fichier
- ou de toute la configuration de votre ordinateur
- à la création du noyau linux (milliers de devs, dizaines de milliers de patches, ...)
- ... en passant par un projet de prog python en M2 ...;-)

# GIT c'est quoi



... utilisé par BEAUCOUP de monde

Et quand je dis beaucoup ...

En 2016, il s'agit du logiciel de gestion de versions le plus populaire qui est utilisé par **plus de douze millions** de personnes dans le monde

c'est pour des projets petits à énormes

- De la sauvegarde des modifs d'UN fichier
- ou de toute la configuration de votre ordinateur
- à la création du noyau linux (milliers de devs, dizaines de milliers de patches, ...)
- ... en passant par un projet de prog python en M2 ...;-)

# GIT c'est quoi



... utilisé par BEAUCOUP de monde

Et quand je dis beaucoup ...

En 2016, il s'agit du logiciel de gestion de versions le plus populaire qui est utilisé par **plus de douze millions** de personnes dans le monde

c'est pour des projets petits à énormes

- De la sauvegarde des modifs d'UN fichier
- ou de toute la configuration de votre ordinateur
- à la création du noyau linux (milliers de devs, dizaines de milliers de patches, ...)
- ... en passant par un projet de prog python en M2 ...;-)

**BREF : tout projet utilisant des fichiers peut utiliser GIT**

# GIT c'est quoi

... et très utile !



OK on peut l'utiliser mais ça sert à quoi EN VRAI ?

- ① À garder l'historique des modifications (voir, revenir et expliquer)

# GIT c'est quoi

... et très utile !



OK on peut l'utiliser mais ça sert à quoi EN VRAI ?

- ➊ À garder l'historique des modifications (voir, revenir et expliquer)
- ➋ À pouvoir faire des tests sans toucher à notre « base qui marche »

# GIT c'est quoi

... et très utile !



OK on peut l'utiliser mais ça sert à quoi EN VRAI ?

- ➊ À garder l'historique des modifications (voir, revenir et expliquer)
- ➋ À pouvoir faire des tests sans toucher à notre « base qui marche »
- ➌ Pouvoir travailler sur plusieurs trucs en même temps

# GIT c'est quoi

... et très utile !



OK on peut l'utiliser mais ça sert à quoi EN VRAI ?

- ➊ À garder l'historique des modifications (voir, revenir et expliquer)
- ➋ À pouvoir faire des tests sans toucher à notre « base qui marche »
- ➌ Pouvoir travailler sur plusieurs trucs en même temps



# GIT c'est quoi



... et très utile !

OK on peut l'utiliser mais ça sert à quoi EN VRAI ?

- ➊ À garder l'historique des modifications (voir, revenir et expliquer)
- ➋ À pouvoir faire des tests sans toucher à notre « base qui marche »
- ➌ Pouvoir travailler sur plusieurs trucs en même temps

---

Mais aussi (pas en local)

- ➍ permettre le travail en parallèle de plusieurs devs (conflits, branches)

# GIT c'est quoi



... et très utile !

OK on peut l'utiliser mais ça sert à quoi EN VRAI ?

- ➊ À garder l'historique des modifications (voir, revenir et expliquer)
- ➋ À pouvoir faire des tests sans toucher à notre « base qui marche »
- ➌ Pouvoir travailler sur plusieurs trucs en même temps

---

Mais aussi (pas en local)

- ➍ permettre le travail en parallèle de plusieurs devs (conflits, branches)
- ➎ permettre de travailler sur son code depuis n'importe quel poste (dev)

# GIT c'est quoi



... et très utile !

OK on peut l'utiliser mais ça sert à quoi EN VRAI ?

- ❶ À garder l'historique des modifications (voir, revenir et expliquer)
- ❷ À pouvoir faire des tests sans toucher à notre « base qui marche »
- ❸ Pouvoir travailler sur plusieurs trucs en même temps

---

Mais aussi (pas en local)

- ❹ permettre le travail en parallèle de plusieurs devs (conflits, branches)
- ❺ permettre de travailler sur son code depuis n'importe quel poste (dev)
- ❻ permettre de faire un accès pour des contributeurs externes (gitHub/GitLab/Savannah, ...)

# Plan

- 1 Planning [Rappels]
- 2 GIT : système de gestion de version
  - Présentation rapide
  - HowTo GIT (Local)
  - Mini TP GIT
  - Autre manière de travailler avec GIT
- 3 Démarrage du projet
  - Projet
  - Évaluation

# Installation

De très nombreux outils existent, en ligne de commande ou graphique  
Pour expliquer on va utiliser l'outil accessible par anaconda et on reviendra sur d'autres outils en fin de présentation de GIT

## Installation

```
conda install -c anaconda git
```

## Test de l'installation

```
user>git --version
```

# Installation

De très nombreux outils existent, en ligne de commande ou graphique  
Pour expliquer on va utiliser l'outil accessible par anaconda et on reviendra sur d'autres outils en fin de présentation de GIT

## Installation

```
conda install -c anaconda git
```

## Test de l'installation

```
user>git --version  
git version 2.43.0
```

# Installation

De très nombreux outils existent, en ligne de commande ou graphique  
Pour expliquer on va utiliser l'outil accessible par anaconda et on reviendra sur d'autres outils en fin de présentation de GIT

## Installation

```
conda install -c anaconda git
```

## Test de l'installation

```
user>git --version  
git version 2.43.0
```

Bravo, vous avez lancé GIT ...;-)

**Et maintenant what ?**

# Plan

- 1 Planning [Rappels]
- 2 **GIT : système de gestion de version**
  - Présentation rapide
  - HowTo GIT (Local)
  - **Mini TP GIT**
  - Autre manière de travailler avec GIT
- 3 Démarrage du projet
  - Projet
  - Évaluation



# Exercice d'application GIT

I/V

Initier un arbre GIT local

## Principe

Pour commencer à travailler avec GIT, il faut **EXPLICITEMENT** créer une hiérarchie GIT → c'est l'initialisation du dépôt (ou arbre) git.

# Exercice d'application GIT

I/V

Initier un arbre GIT local

## Principe

Pour commencer à travailler avec GIT, il faut **EXPLICITEMENT** créer une hiérarchie GIT → c'est l'initialisation du dépôt (ou arbre) git.

## Commandes

```
user> git init test (en local)
```

# Exercice d'application GIT

I/V

Initier un arbre GIT local

## Principe

Pour commencer à travailler avec GIT, il faut **EXPLICITEMENT** créer une hiérarchie GIT → c'est l'initialisation du dépôt (ou arbre) git.

## Commandes

```
user> git init test (en local)
```

```
Initialized empty Git repository in [...]/test/.git/
```

# Exercice d'application GIT

II/V

Faire un texte basique + commiter

## Principe

- 1 GIT ne suit pas automatiquement les fichiers,

# Exercice d'application GIT

II/V

Faire un texte basique + commiter

## Principe

- 1 GIT ne suit pas automatiquement les fichiers,
  - on doit lui indiquer les fichiers à suivre,

# Exercice d'application GIT

II/V

Faire un texte basique + commiter

## Principe

- 1 GIT ne suit pas automatiquement les fichiers,
  - on doit lui indiquer les fichiers à suivre,
  - et quand en prendre une nouvelle version !

# Exercice d'application GIT

II/V

Faire un texte basique + commiter

## Principe

- ① GIT ne suit pas automatiquement les fichiers,
  - on doit lui indiquer les fichiers à suivre,
  - et quand en prendre une nouvelle version !
- ② il existe 2 zones : travail (staging) et stockage (commit)

# Exercice d'application GIT

II/V

Faire un texte basique + commiter

## Principe

- ① GIT ne suit pas automatiquement les fichiers,
  - on doit lui indiquer les fichiers à suivre,
  - et quand en prendre une nouvelle version !
- ② il existe 2 zones : travail (staging) et stockage (commit)



# Exercice d'application GIT

II/V

Faire un texte basique + commiter

## Principe

- 1 GIT ne suit pas automatiquement les fichiers,
  - on doit lui indiquer les fichiers à suivre,
  - et quand en prendre une nouvelle version !
- 2 il existe 2 zones : travail (staging) et stockage (commit)

Un peu d'explications

**staging** c'est une zone temporaire où vous allez mettre la série de modification pendant que vous travaillez

# Exercice d'application GIT

II/V

Faire un texte basique + commiter

## Principe

- 1 GIT ne suit pas automatiquement les fichiers,
  - on doit lui indiquer les fichiers à suivre,
  - et quand en prendre une nouvelle version !
- 2 il existe 2 zones : travail (staging) et stockage (commit)

Un peu d'explications

**staging** c'est une zone temporaire où vous allez mettre la série de modification pendant que vous travaillez

**commit** c'est la zone de stockage qui garde la/les version(s) voulue(s) des fichiers suivis et modifiés

Et maintenant passons à la pratique !

# Exercice d'application GIT

II/V

Faire un texte basique + commiter

## Commandes

- Créer un fichier texte fichier.txt dans le répertoire GIT (test)

# Exercice d'application GIT

II/V

Faire un texte basique + commiter

## Commandes

- Créer un fichier texte fichier.txt dans le répertoire GIT (test)
- GIT peut reconnaître un truc non suivi avec `user> git status`

# Exercice d'application GIT

II/V

Faire un texte basique + commiter

## Commandes

- Créer un fichier texte fichier.txt dans le répertoire GIT (test)
- GIT peut reconnaître un truc non suivi avec `user> git status`

On branch master

No commits yet

Untracked files:

(use "`git add <file>...`" to include in what will be committed)  
fichier.txt

nothing added to commit but untracked files present (use "`git add`" to track)

# Exercice d'application GIT

II/V

Faire un texte basique + commiter

## Commandes

- Créer un fichier texte fichier.txt dans le répertoire GIT (test)
- GIT peut reconnaître un truc non suivi avec `user> git status`
- Ajouter le à la branche master avec `user> git add fichier.txt`  
puis `user> git status` again

# Exercice d'application GIT

II/V

Faire un texte basique + commiter

## Commandes

- Créer un fichier texte fichier.txt dans le répertoire GIT (test)
- GIT peut reconnaître un truc non suivi avec `user> git status`
- Ajouter le à la branche master avec `user> git add fichier.txt`  
puis `user> git status` again

On branch master

No commits yet

Changes to be committed:

(use "`git rm --cached <file>...`" to unstage)  
new file: fichier.txt

# Exercice d'application GIT

II/V

Faire un texte basique + commiter

## Commandes

- Créer un fichier texte fichier.txt dans le répertoire GIT (test)
- GIT peut reconnaître un truc non suivi avec `user> git status`
- Ajouter le à la branche master avec `user> git add fichier.txt`  
puis `user> git status` again
- Commiter le fichier avec `user> git commit -m "Initial"`



# Exercice d'application GIT

II/V

Faire un texte basique + commiter

## Commandes

- Créer un fichier texte fichier.txt dans le répertoire GIT (test)
- GIT peut reconnaître un truc non suivi avec `user> git status`
- Ajouter le à la branche master avec `user> git add fichier.txt`  
puis `user> git status` again
- Commiter le fichier avec `user> git commit -m "Initial"`

```
[master (root-commit) 1a17ed8] Initial
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 fichier.txt
```

# Exercice d'application GIT

II/V

Faire un texte basique + commiter

## Commandes

- Créer un fichier texte fichier.txt dans le répertoire GIT (test)
- GIT peut reconnaître un truc non suivi avec `user> git status`
- Ajouter le à la branche master avec `user> git add fichier.txt`  
puis `user> git status` again
- Commiter le fichier avec `user> git commit -m "Initial"`
- Vérifier le statut avec `user> git status`

# Exercice d'application GIT

II/V

Faire un texte basique + commiter

## Commandes

- Créer un fichier texte fichier.txt dans le répertoire GIT (test)
- GIT peut reconnaître un truc non suivi avec `user> git status`
- Ajouter le à la branche master avec `user> git add fichier.txt`  
puis `user> git status` again
- Commiter le fichier avec `user> git commit -m "Initial"`
- Vérifier le statut avec `user> git status`

On branch master

nothing to commit, working tree clean

# Exercice d'application GIT

III/V

Faire une modif + voir les diffs + commit(2)

## Principe

Nous avons maintenant le fichier `fichier.txt` qui est suivi par GIT !

Faisons maintenant une modification et utilisons git pour voir la modification puis valider !

# Exercice d'application GIT

III/V

Faire une modif + voir les diffs + commit(2)

## Commandes

- Ajouter la ligne « Texte ajouté » au fichier `fichier.txt` `fichier.txt`

# Exercice d'application GIT

III/V

Faire une modif + voir les diffs + commit(2)

## Commandes

- Ajouter la ligne « Texte ajouté » au fichier fichier.txt
- Une petite vérif permet de voir qu'il est modifié (user> git status)

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)  
modified: fichier.txt

no changes added to commit (use "git add" and/or "git commit -a")

# Exercice d'application GIT

III/V

Faire une modif + voir les diffs + commit(2)

## Commandes

- Ajouter la ligne « Texte ajouté » au fichier fichier.txt `fichier.txt`
- Une petite vérif permet de voir qu'il est modifié (`user> git status`)
- On peut voir les modifs avec `user> git diff fichier.txt`

```
diff --git a/fichier.txt b/fichier.txt
index e69de29..320ff41 100644
--- a/fichier.txt
+++ b/fichier.txt
@@ -0,0 +1 @@
+Texte ajouté
```

# Exercice d'application GIT

III/V

Faire une modif + voir les diffs + commit(2)

## Commandes

- Ajouter la ligne « Texte ajouté » au fichier fichier.txt `fichier.txt`
- Une petite vérif permet de voir qu'il est modifié (`user> git status`)
- On peut voir les modifs avec `user> git diff fichier.txt`
- Commit du fichier : `user> git commit -a -m "Ajout de ligne"`



# Exercice d'application GIT

III/V

Faire une modif + voir les diffs + commit(2)

## Commandes

- Ajouter la ligne « Texte ajouté » au fichier fichier.txt
- Une petite vérif permet de voir qu'il est modifié (user> git status)
- On peut voir les mods avec user> git diff fichier.txt
- Commit du fichier : user> git commit -a -m "Ajout de ligne"
- On peut voir la suite de mise à jour à l'aide la commande user> git log)

```
commit 4c0b291b70ffff3a2ba5759dfc5e2423f3e8815d (HEAD -> master)
Author: Clement <clement.baty@wanadoo.fr>
Date:   Mon Dec 18 09:25:36 2023 +0100
```

Ajout de ligne

```
commit ac4e8e1eb03e572b6356490da0ec7359f02baf21
Author: Clement <clement.baty@wanadoo.fr>
Date:   Mon Dec 18 09:24:09 2023 +0100
```

Initial

# Exercice d'application GIT

IV/V

## Faire une branche

### Principe

Faire une nouvelle branche permet de stocker des fichiers qu'on pense ajouter mais sans que notre branche principale soit impactée.

# Exercice d'application GIT

IV/V

## Faire une branche

### Principe

Faire une nouvelle branche permet de stocker des fichiers qu'on pense ajouter mais sans que notre branche principale soit impactée. Ce qui est encore mieux c'est de pouvoir faire fusionner notre nouvelle branche avec notre tronc (branche master) !

# Exercice d'application GIT

IV/V

## Faire une branche

### Principe

Faire une nouvelle branche permet de stocker des fichiers qu'on pense ajouter mais sans que notre branche principale soit impactée. Ce qui est encore mieux c'est de pouvoir faire fusionner notre nouvelle branche avec notre tronc (branche master) !

- On va créer une nouvelle branche « essais » et s'y rendre

# Exercice d'application GIT

IV/V

## Faire une branche

### Principe

Faire une nouvelle branche permet de stocker des fichiers qu'on pense ajouter mais sans que notre branche principale soit impactée. Ce qui est encore mieux c'est de pouvoir faire fusionner notre nouvelle branche avec notre tronc (branche master) !

- On va créer une nouvelle branche « essais » et s'y rendre
- Vérifier qu'on est bien dans la bonne branche

# Exercice d'application GIT

IV/V

## Faire une branche

### Principe

Faire une nouvelle branche permet de stocker des fichiers qu'on pense ajouter mais sans que notre branche principale soit impactée.

Ce qui est encore mieux c'est de pouvoir faire fusionner notre nouvelle branche avec notre tronc (branche master) !

- On va créer une nouvelle branche « essais » et s'y rendre
- Vérifier qu'on est bien dans la bonne branche
- On va créer un fichier `file.txt` dans la branche « essais » et l'ajouter

# Exercice d'application GIT

IV/V

## Faire une branche

### Principe

Faire une nouvelle branche permet de stocker des fichiers qu'on pense ajouter mais sans que notre branche principale soit impactée.

Ce qui est encore mieux c'est de pouvoir faire fusionner notre nouvelle branche avec notre tronc (branche master) !

- On va créer une nouvelle branche « essais » et s'y rendre
- Vérifier qu'on est bien dans la bonne branche
- On va créer un fichier `file.txt` dans la branche « essais » et l'ajouter
- On va modifier le fichier `file.txt` et sauvegarder la modif

# Exercice d'application GIT

IV/V

## Faire une branche

### Principe

Faire une nouvelle branche permet de stocker des fichiers qu'on pense ajouter mais sans que notre branche principale soit impactée.

Ce qui est encore mieux c'est de pouvoir faire fusionner notre nouvelle branche avec notre tronc (branche master) !

- On va créer une nouvelle branche « essais » et s'y rendre
- Vérifier qu'on est bien dans la bonne branche
- On va créer un fichier `file.txt` dans la branche « essais » et l'ajouter
- On va modifier le fichier `file.txt` et sauvegarder la modif
- On va vérifier que notre branche est clean



# Exercice d'application GIT

IV/V

## Faire une branche

### Principe

Faire une nouvelle branche permet de stocker des fichiers qu'on pense ajouter mais sans que notre branche principale soit impactée.

Ce qui est encore mieux c'est de pouvoir faire fusionner notre nouvelle branche avec notre tronc (branche master) !

- On va créer une nouvelle branche « essais » et s'y rendre
- Vérifier qu'on est bien dans la bonne branche
- On va créer un fichier `file.txt` dans la branche « essais » et l'ajouter
- On va modifier le fichier `file.txt` et sauvegarder la modif
- On va vérifier que notre branche est clean
- On va merger le tronc (branche principale) et la nouvelle branche

# Exercice d'application GIT

IV/V

## Faire une branche

### Principe

Faire une nouvelle branche permet de stocker des fichiers qu'on pense ajouter mais sans que notre branche principale soit impactée. Ce qui est encore mieux c'est de pouvoir faire fusionner notre nouvelle branche avec notre tronc (branche master) !

- On va créer une nouvelle branche « essais » et s'y rendre
- Vérifier qu'on est bien dans la bonne branche
- On va créer un fichier `file.txt` dans la branche « essais » et l'ajouter
- On va modifier le fichier `file.txt` et sauvegarder la modif
- On va vérifier que notre branche est clean
- On va merger le tronc (branche principale) et la nouvelle branche
- Et on vérifie que tout va bien :)

# Exercice d'application GIT

IV/V

Faire une branche

## Commandes

- Création branche « essais » : `user> git branch essais` puis s'y rendre avec `user> git checkout essais`
- Vérifier qu'on est bien dans la bonne branche `user> git branch`

# Exercice d'application GIT

IV/V

Faire une branche

## Commandes

- Création branche « essais » : `user> git branch essais` puis s'y rendre avec `user> git checkout essais`
- Vérifier qu'on est bien dans la bonne branche `user> git branch`
  - \* `essais`
  - `master`

# Exercice d'application GIT

IV/V

Faire une branche

## Commandes

- Création branche « essais » : `user> git branch essais` puis s'y rendre avec `user> git checkout essais`
- Vérifier qu'on est bien dans la bonne branche `user> git branch`
- Création du fichier `file.txt` avec seulement le texte « Bonjour NAME » puis `user> git add file.txt` puis `user> git commit -a -m " MAJ dépôt "`

# Exercice d'application GIT

IV/V

Faire une branche

## Commandes

- Création branche « essais » : `user> git branch essais` puis s'y rendre avec `user> git checkout essais`
- Vérifier qu'on est bien dans la bonne branche `user> git branch`
- Création du fichier `file.txt` avec seulement le texte « Bonjour NAME » puis `user> git add file.txt` puis `user> git commit -a -m "MAJ dépôt"`
- Modification du fichier `file.txt` pour obtenir « Bonjour **mon ami** NAME » puis `user> git commit -a -m "Ajout de mon ami"`

# Exercice d'application GIT

IV/V

Faire une branche

## Commandes

- Création branche « essais » : `user> git branch essais` puis s'y rendre avec `user> git checkout essais`
- Vérifier qu'on est bien dans la bonne branche `user> git branch`
- Création du fichier `file.txt` avec seulement le texte « Bonjour NAME » puis `user> git add file.txt` puis `user> git commit -a -m " MAJ dépôt"`
- Modification du fichier `file.txt` pour obtenir « Bonjour **mon ami** NAME » puis `user> git commit -a -m "Ajout de mon ami"`
- Vérification du status clean de la branche avec `user> git status`

# Exercice d'application GIT

IV/V

Faire une branche

## Commandes

- Création branche « essais » : `user> git branch essais` puis s'y rendre avec `user> git checkout essais`
- Vérifier qu'on est bien dans la bonne branche `user> git branch`
- Création du fichier `file.txt` avec seulement le texte « Bonjour NAME » puis `user> git add file.txt` puis `user> git commit -a -m " MAJ dépôt"`
- Modification du fichier `file.txt` pour obtenir « Bonjour **mon ami** NAME » puis `user> git commit -a -m "Ajout de mon ami"`
- Vérification du status clean de la branche avec `user> git status`  
On branch essais  
nothing to commit, working tree clean



# Exercice d'application GIT

IV/V

Faire une branche

## Commandes

- Création branche « essais » : `user> git branch essais` puis s'y rendre avec `user> git checkout essais`
- Vérifier qu'on est bien dans la bonne branche `user> git branch`
- Création du fichier `file.txt` avec seulement le texte « Bonjour NAME » puis `user> git add file.txt` puis `user> git commit -a -m " MAJ dépôt"`
- Modification du fichier `file.txt` pour obtenir « Bonjour **mon ami** NAME » puis `user> git commit -a -m "Ajout de mon ami"`
- Vérification du status clean de la branche avec `user> git status`
- Mergeons (pardon fusionnons) maintenant la branche « essais » avec master : `user> git checkout master` puis `user> git merge essais`

# Exercice d'application GIT

IV/V

## Faire une branche

### Commandes

- Création branche « essais » : `user> git branch essais` puis s'y rendre avec `user> git checkout essais`
- Vérifier qu'on est bien dans la bonne branche `user> git branch`
- Création du fichier `file.txt` avec seulement le texte « Bonjour NAME » puis `user> git add file.txt` puis `user> git commit -a -m " MAJ dépôt"`
- Modification du fichier `file.txt` pour obtenir « Bonjour **mon ami** NAME » puis `user> git commit -a -m "Ajout de mon ami"`
- Vérification du status clean de la branche avec `user> git status`
- Mergeons (pardon fusionnons) maintenant la branche « essais » avec master :  
`user> git checkout master` puis `user> git merge essais`  
Updating dcf2b4e..7e1580c  
Fast-forward  
file.txt | 1 +  
1 file changed, 1 insertion(+)  
create mode 100644 file.txt

# Exercice d'application GIT

V/V

Aller plus loin

## Pour aller plus loin

- l'aide en ligne directe

# Exercice d'application GIT

V/V

Aller plus loin

## Pour aller plus loin

- l'aide en ligne directe
- l'aide en ligne demandée

# Exercice d'application GIT

V/V

Aller plus loin

## Pour aller plus loin

- l'aide en ligne directe
- l'aide en ligne demandée
- l'aide sur internet par exemple

`openclassrooms.com/fr/courses/1233741-gerez-vos-codes-source-avec-git`

---

# Exercice d'application GIT

V/V

Aller plus loin

## Pour aller plus loin

- l'aide en ligne directe
- l'aide en ligne demandée
- l'aide sur internet par exemple

`openclassrooms.com/fr/courses/1233741-gerez-vos-codes-source-avec-git`

---

- et surtout, il faut pratiquer, après le cycle classique est :

# Exercice d'application GIT

V/V

Aller plus loin

## Pour aller plus loin

- l'aide en ligne directe
- l'aide en ligne demandée
- l'aide sur internet par exemple

[openclassrooms.com/fr/courses/1233741-gerez-vos-codes-source-avec-git](https://openclassrooms.com/fr/courses/1233741-gerez-vos-codes-source-avec-git)

---

- et surtout, il faut pratiquer, après le cycle classique est :
  - 1 modifier le fichier normalement

# Exercice d'application GIT

V/V

Aller plus loin

## Pour aller plus loin

- l'aide en ligne directe
- l'aide en ligne demandée
- l'aide sur internet par exemple

[openclassrooms.com/fr/courses/1233741-gerez-vos-codes-source-avec-git](https://openclassrooms.com/fr/courses/1233741-gerez-vos-codes-source-avec-git)

- 
- et surtout, il faut pratiquer, après le cycle classique est :
    - 1 modifier le fichier normalement
    - 2 `user>git add file` (indexer les modifs fichier, nécessaire à la création du fichier)



# Exercice d'application GIT

V/V

Aller plus loin

## Pour aller plus loin

- l'aide en ligne directe
- l'aide en ligne demandée
- l'aide sur internet par exemple

[openclassrooms.com/fr/courses/1233741-gerez-vos-codes-source-avec-git](https://openclassrooms.com/fr/courses/1233741-gerez-vos-codes-source-avec-git)

- 
- et surtout, il faut pratiquer, après le cycle classique est :
    - 1 modifier le fichier normalement
    - 2 `user>git add file` (indexer les modifs fichier, nécessaire à la création du fichier)
    - 3 `user>git commit -a -m "Message de commit"` (valider les version actuelles des fichiers modifiés)

# Exercice d'application GIT

V/V

Aller plus loin

## Pour aller plus loin

- l'aide en ligne directe
- l'aide en ligne demandée
- l'aide sur internet par exemple

[openclassrooms.com/fr/courses/1233741-gerez-vos-codes-source-avec-git](https://openclassrooms.com/fr/courses/1233741-gerez-vos-codes-source-avec-git)

- 
- et surtout, il faut pratiquer, après le cycle classique est :
    - ① modifier le fichier normalement
    - ② `user>git add file` (indexer les modifs fichier, nécessaire à la création du fichier)
    - ③ `user>git commit -a -m "Message de commit"` (valider les version actuelles des fichiers modifiés)
    - ④ `user>git status` pour voir l'état de la copie locale

# Exercice d'application GIT

V/V

Aller plus loin

## Pour aller plus loin

- l'aide en ligne directe
- l'aide en ligne demandée
- l'aide sur internet par exemple

[openclassrooms.com/fr/courses/1233741-gerez-vos-codes-source-avec-git](https://openclassrooms.com/fr/courses/1233741-gerez-vos-codes-source-avec-git)

- 
- et surtout, il faut pratiquer, après le cycle classique est :
    - ① modifier le fichier normalement
    - ② `user>git add file` (indexer les modifs fichier, nécessaire à la création du fichier)
    - ③ `user>git commit -a -m "Message de commit"` (valider les version actuelles des fichiers modifiés)
    - ④ `user>git status` pour voir l'état de la copie locale
    - ⑤ `user>gitk` pour l'historique des commits

# Plan

- 1 Planning [Rappels]
- 2 **GIT : système de gestion de version**
  - Présentation rapide
  - HowTo GIT (Local)
  - Mini TP GIT
  - **Autre manière de travailler avec GIT**
- 3 Démarrage du projet
  - Projet
  - Évaluation

# Autres outils GIT

I/IV

Voir l'arbre des modifs + récupérer une ancienne version

## Principe

Bon maintenant que vous avez vu le cœur de GIT voyons des outils sympas (et parfois déjà installés)

① gitk (-all)

# Autres outils GIT

I/IV

Voir l'arbre des modifs + récupérer une ancienne version

## Principe

Bon maintenant que vous avez vu le cœur de GIT voyons des outils sympas (et parfois déjà installés)

- 1 gitk (-all)
- 2 autres outils

# Autres outils GIT

I/IV

Voir l'arbre des modifs + récupérer une ancienne version

## Principe

Bon maintenant que vous avez vu le cœur de GIT voyons des outils sympas (et parfois déjà installés)

- 1 gitk (-all)
- 2 autres outils

# Autres outils GIT

I/IV

Voir l'arbre des modifs + récupérer une ancienne version

## Principe

Bon maintenant que vous avez vu le cœur de GIT voyons des outils sympas (et parfois déjà installés)

- 1 gitk (-all)
- 2 autres outils

## Commandes

- 1 gitk se lance en faisant `user> gitk --all`



# Autres outils GIT

I/IV

Voir l'arbre des modifs + récupérer une ancienne version

## Principe

Bon maintenant que vous avez vu le cœur de GIT voyons des outils sympas (et parfois déjà installés)

- 1 gitk (-all)
- 2 autres outils

## Commandes

- 1 gitk se lance en faisant `user> gitk --all`
  - 1 on peut voir/créer/merger des branches

# Autres outils GIT

I/IV

Voir l'arbre des modifs + récupérer une ancienne version

## Principe

Bon maintenant que vous avez vu le cœur de GIT voyons des outils sympas (et parfois déjà installés)

- 1 gitk (-all)
- 2 autres outils

## Commandes

- 1 gitk se lance en faisant `user> gitk --all`
  - 1 on peut voir/créer/merger des branches
  - 2 on peut supprimer/amender des commits

# Autres outils GIT

I/IV

Voir l'arbre des modifs + récupérer une ancienne version

## Principe

Bon maintenant que vous avez vu le cœur de GIT voyons des outils sympas (et parfois déjà installés)

- 1 gitk (-all)
- 2 autres outils

## Commandes

- 1 gitk se lance en faisant `user> gitk --all`
  - 1 on peut voir/créer/merger des branches
  - 2 on peut supprimer/amender des commits
  - 3 on peut démarrer une fenêtre graphique détaillant les modifs

# Autres outils GIT

I/IV

Voir l'arbre des modifs + récupérer une ancienne version

## Principe

Bon maintenant que vous avez vu le cœur de GIT voyons des outils sympas (et parfois déjà installés)

- 1 gitk (-all)
- 2 autres outils

## Commandes

- 1 gitk se lance en faisant `user> gitk --all`
  - 1 on peut voir/créer/merger des branches
  - 2 on peut supprimer/amender des commits
  - 3 on peut démarrer une fenêtre graphique détaillant les modifs
- 2 des applications graphiques dédiées :

# Autres outils GIT

I/IV

Voir l'arbre des modifs + récupérer une ancienne version

## Principe

Bon maintenant que vous avez vu le cœur de GIT voyons des outils sympas (et parfois déjà installés)

- 1 gitk (-all)
- 2 autres outils

## Commandes

- 1 gitk se lance en faisant `user> gitk --all`
  - 1 on peut voir/créer/merger des branches
  - 2 on peut supprimer/amender des commits
  - 3 on peut démarrer une fenêtre graphique détaillant les modifs
- 2 des applications graphiques dédiées :
  - <https://git-scm.com/download/win>

# Autres outils GIT

I/IV

Voir l'arbre des modifs + récupérer une ancienne version

## Principe

Bon maintenant que vous avez vu le cœur de GIT voyons des outils sympas (et parfois déjà installés)

- 1 gitk (-all)
- 2 autres outils

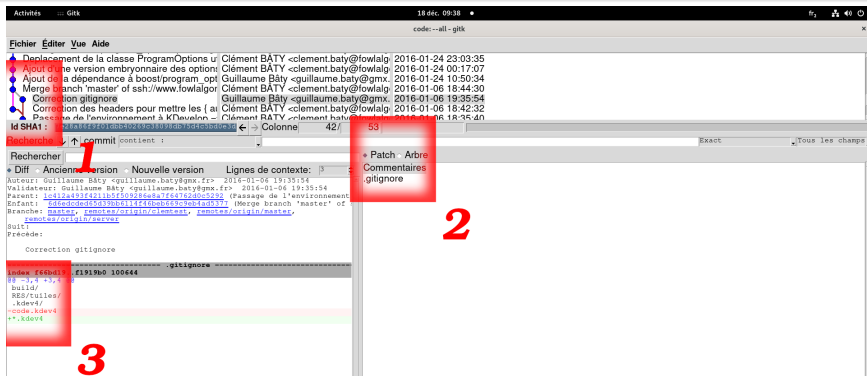
## Commandes

- 1 gitk se lance en faisant `user> gitk --all`
  - 1 on peut voir/créer/merger des branches
  - 2 on peut supprimer/amender des commits
  - 3 on peut démarrer une fenêtre graphique détaillant les modifs
- 2 des applications graphiques dédiées :
  - <https://git-scm.com/download/win>
  - <https://tortoisegit.org/>

## Autres outils GIT

II/IV

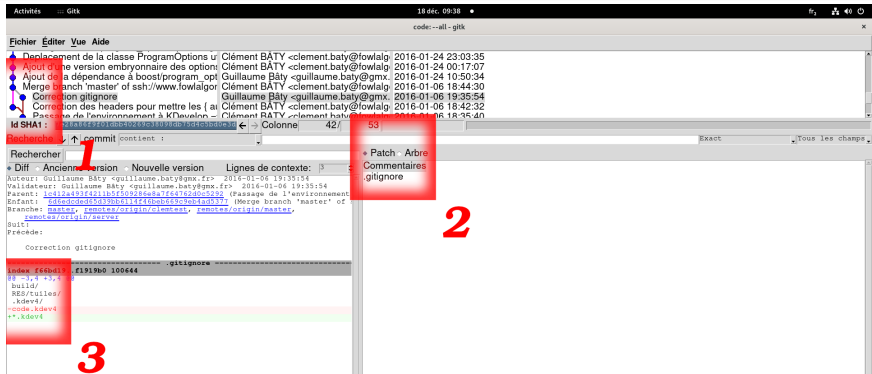
## Détail de la fenêtre GTK



# Autres outils GIT

II/IV

## Détail de la fenêtre GITK



## Description rapide

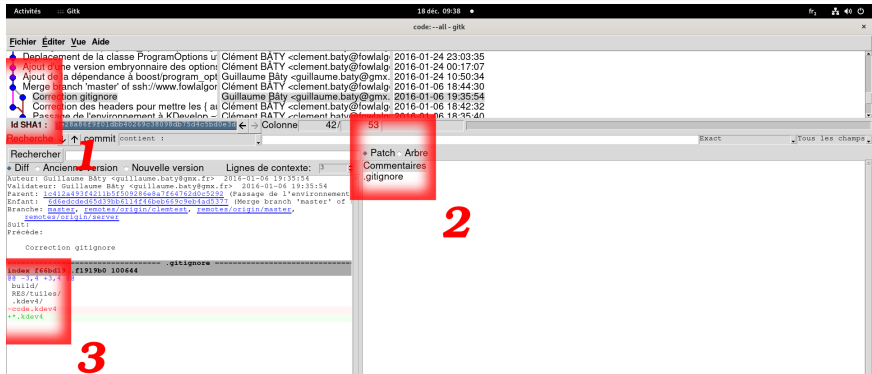
### ① Un merge de branche



# Autres outils GIT

II/IV

## Détail de la fenêtre GITK



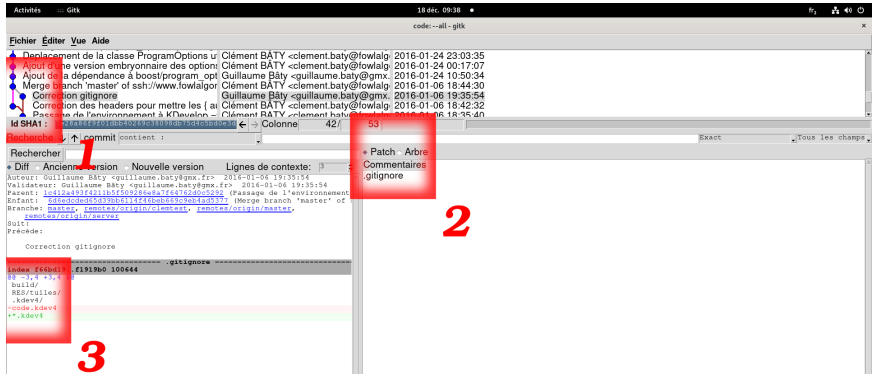
## Description rapide

- ① Un merge de branche
- ② Un résumé des fichiers qui ont changés

# Autres outils GIT

II/IV

## Détail de la fenêtre GITK



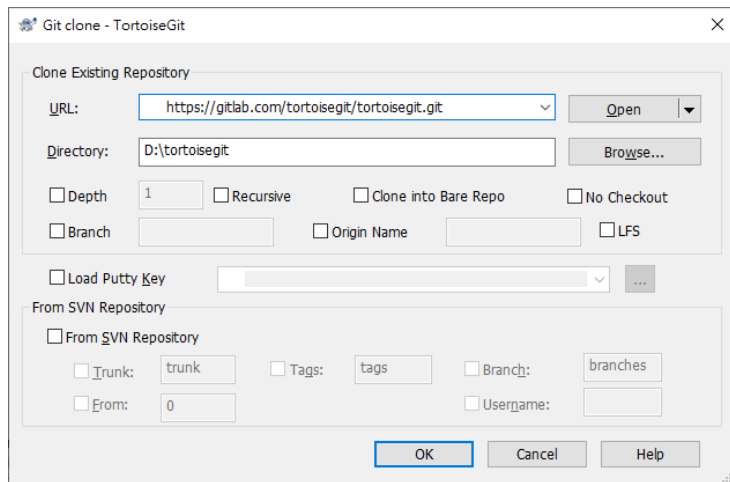
## Description rapide

- 1 Un merge de branche
- 2 Un résumé des fichiers qui ont changés
- 3 le fichier .gitignore

# Autres outils GIT

III/IV

## Détail de TortoiseGit

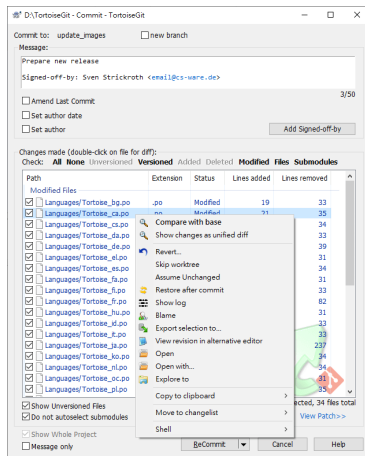


Git clone

# Autres outils GIT

III/IV

## Détail de TortoiseGit



Git commit

## Autres outils GIT

III/IV

## Détail de TortoiseGit

D:\TortoiseGit - Log Messages - TortoiseGit

update\_images From: 2008-11-05 To: 2020-03-16 / Pati Author Email

Graph	SHA-1	Actions	Message	Author	Date
	331205cf...		Merge branch 'performance'	Sven Strickroth	2020-02-21 02:55:49
	471ae09...		No need to empty CString	Sven Strickroth	2020-02-18 04:14:23
	96154d2d...		Handle special case only if relevant	Sven Strickroth	2020-02-15 21:26:14
	bbd9272a...		Reduce number of CString intermedia...	Sven Strickroth	2020-02-15 19:56:07
	77d7fa0b...		Use CString to shorten conversions	Sven Strickroth	2020-02-15 06:09:51
	a254577f...		Use GetUnicodeLength	Sven Strickroth	2020-02-14 04:29:53
	ef02ddae...		Merge branch 'unicodeutils'	Sven Strickroth	2020-02-16 05:36:54
	8ecb4879...		Prevent implicit casts to CStringA	Sven Strickroth	2020-02-14 02:03:33
	4e51352b...		Reduce buffer sizes	Sven Strickroth	2020-02-14 01:57:24
	c7df4027...		Optimize for small strings	Sven Strickroth	2020-02-14 01:21:54
	b89c0358...		Inline wrapper method	Sven Strickroth	2020-02-13 01:52:04
	22157225...		Add unit test for GetCopyrightForSelf	Sven Strickroth	2020-02-12 04:41:48

SHA-1: 471ae0999ea83dd5e5fcc503ac84d60e066445f  
Describe: REL\_2.9.0.0\_EXTERNAL-209-g471ae0

\* No need to empty CString

Signed-off-by: Sven Strickroth <email@cs-ware.de>

Path	Extension	Status	Lines added	Lines removed
src/TortoiseGitBlame/TortoiseGitBlameData.cpp	.cpp	Modified	0	2

Showing 10430 revision(s), from revision 60636b39 to revision c643f89f - 1 revision(s) selected, 0 file(s) selected; line: 0(+) 2(-)  
files modified = 1 added = 0 deleted = 0 renamed = 0

☒ Show Whole Project

☐ All Branches

Filter paths

Help

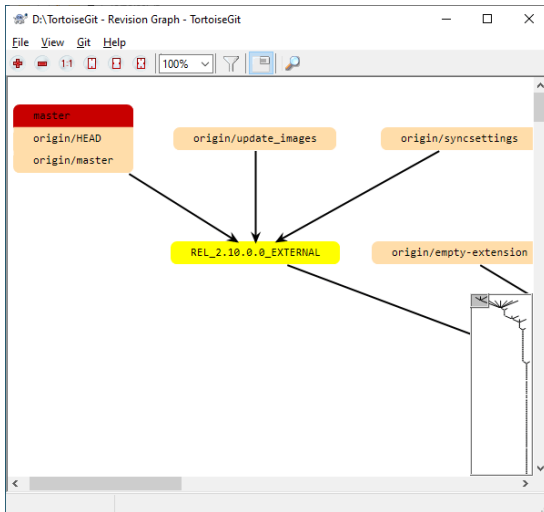
Refresh Statistics Walk Behaviour View OK

## Git log

# Autres outils GIT

III/IV

## Détail de TortoiseGit



Revisions graphs

# Autres outils GIT

IV/IV

## Détail de GitHub/savanah/Gitlab

The screenshot shows a web browser displaying the GitHub repository page for 'AUTOTUNE-UJM' by user 'samm92985'. The repository is private. The page shows the repository name, a search bar, and navigation links like Code, Issues, Pull requests, Actions, Projects, Security, and Insights. Below the repository name, there are buttons for Watch (1), Fork (0), and Star (0). The main content area shows a list of files under the 'master' branch, including 'CODE Traitement de signal samibenticha.py', 'au.wav', 'code\_app\_samibenticha.py', 'm.wav', 'samibenticha.ui', and 'samibenticha\_ui.py'. Each file has a link to 'Add files via upload' and a timestamp of '2 years ago'. There is also a '2 Commits' link. On the right side, there are sections for 'About' (Project autotune), 'Releases' (No releases published), and 'Packages' (No packages published).

Activities Firefox 18 déc. 10:32

Fichier Édition Affichage Historique Marque-pages Outils Aide

samm92985/AUTOTUNE-UJM

Collectif 88 % - ENSE... Pad Prise de notes Enn... planning l'ennemi de l'... Dalle-3 BING 7 DLG

Navigation privée

Autres marque-pages

Q Type to search

<> Code Issues Pull requests Actions Projects Security Insights

AUTOTUNE-UJM Private

Watch 1 Fork 0 Star 0

master 1 Branch 0 Tags

Go to file Add file Code

About

Projet autotune

Activity

0 stars

1 watching

0 forks

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

File	Action	Time
CODE Traitement de signal samibenticha.py	Add files via upload	2 years ago
au.wav	Add files via upload	2 years ago
code_app_samibenticha.py	Add files via upload	2 years ago
m.wav	Add files via upload	2 years ago
samibenticha.ui	Add files via upload	2 years ago
samibenticha_ui.py	Add files via upload	2 years ago

README

## GitHub accueil

# Autres outils GIT

IV/IV

## Détail de GitHub/savanah/Gitlab

Activities Firefox 18 déc. 10:32

Fichier Édition Affichage Historique Marque-pages Outils Aide

samm92985/AUTOTUNE-UJM

Collectif 88 % - ENSE... Pad Prise de notes Enn... planning l'ennemi de l'... Dalle-3 BING 7 DLG

Navigation privée

Autres marque-pages

Code Issues Pull requests Actions Projects Security Insights

AUTOTUNE-UJM Private

Watch 1 Fork 0 Star 0

master 1 Branch 0 Tags

Go to file Add file Code

Local Codespaces

Clone

HTTPS SSH GitHub CLI

https://github.com/samm92985/AUTOTUNE-UJM.git

Use Git or checkout with SVN using the web URL.

Download ZIP

2 years ago

README

About

Projet autotune

Activity

0 stars

1 watching

0 forks

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

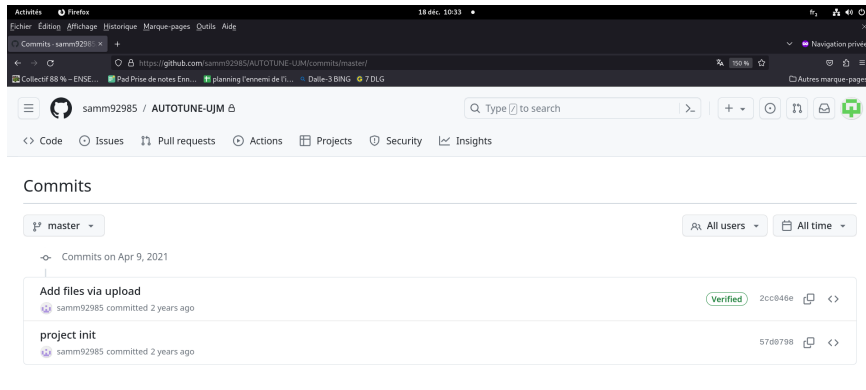
### Git clone



# Autres outils GIT

IV/IV

## Détail de GitHub/savanah/Gitlab



The screenshot shows a web browser window displaying the GitHub repository page for `samm92985 / AUTOTUNE-UJM`. The page is viewed on a desktop, with the browser's address bar showing the URL `https://github.com/samm92985/AUTOTUNE-UJM/commits/master`. The repository's navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Security, and Insights. The main content area is titled "Commits" and shows a list of commits for the `master` branch. The first commit is titled "Add files via upload" and was made by `samm92985` 2 years ago. The second commit is titled "project init" and was also made by `samm92985` 2 years ago. The page includes filters for "All users" and "All time".

Commits

master

Commits on Apr 9, 2021

Add files via upload Verified 2cc046e

samm92985 committed 2 years ago

project init 57d0798

samm92985 committed 2 years ago

## Git log

# Plan

- 1 Planning [Rappels]
- 2 GIT : système de gestion de version
  - Présentation rapide
  - HowTo GIT (Local)
  - Mini TP GIT
  - Autre manière de travailler avec GIT
- 3 Démarrage du projet
  - Projet
  - Évaluation

# Plan

- 1 Planning [Rappels]
- 2 GIT : système de gestion de version
  - Présentation rapide
  - HowTo GIT (Local)
  - Mini TP GIT
  - Autre manière de travailler avec GIT
- 3 Démarrage du projet
  - **Projet**
  - Évaluation

# Description basique du projet

Et le projet c'est ...

Une analyse et modélisation dynamique de la consommation électrique et de gaz des français !

# Description basique du projet

Et le projet c'est ...

Une analyse et modélisation dynamique de la consommation électrique et de gaz des français !

**Dataset utilisé** (OpenData officiel et mis à jour régulièrement)

<https://odre.opendatasoft.com/explore/dataset/consommation-quotidienne-brute>

# Description basique du projet

## Et le projet c'est ...

Une analyse et modélisation dynamique de la consommation électrique et de gaz des français !

**Dataset utilisé** (OpenData officiel et mis à jour régulièrement)

<https://odre.opendatasoft.com/explore/dataset/consummation-quotidienne-brute>

## Euh c'est à dire

- 1 On laisse la possibilité de récupérer dynamiquement depuis internet mais aussi d'un fichier local choisi par l'utilisateur

# Description basique du projet

## Et le projet c'est ...

Une analyse et modélisation dynamique de la consommation électrique et de gaz des français !

**Dataset utilisé** (OpenData officiel et mis à jour régulièrement)

<https://odre.opendatasoft.com/explore/dataset/consommation-quotidienne-brute>

## Euh c'est à dire

- ① On laisse la possibilité de récupérer dynamiquement depuis internet mais aussi d'un fichier local choisi par l'utilisateur
- ② On demande à l'utilisateur ce qu'il veut savoir sur ce dataset :

# Description basique du projet

## Et le projet c'est ...

Une analyse et modélisation dynamique de la consommation électrique et de gaz des français !

**Dataset utilisé** (OpenData officiel et mis à jour régulièrement)

<https://odre.opendatasoft.com/explore/dataset/consommation-quotidienne-brute>

## Euh c'est à dire

- ① On laisse la possibilité de récupérer dynamiquement depuis internet mais aussi d'un fichier local choisi par l'utilisateur
- ② On demande à l'utilisateur ce qu'il veut savoir sur ce dataset :
  - le minimum, le maximum, valeur moyenne sur une plage donnée



# Description basique du projet

## Et le projet c'est ...

Une analyse et modélisation dynamique de la consommation électrique et de gaz des français !

**Dataset utilisé** (OpenData officiel et mis à jour régulièrement)

<https://odre.opendatasoft.com/explore/dataset/consommation-quotidienne-brute>

## Euh c'est à dire

- ① On laisse la possibilité de récupérer dynamiquement depuis internet mais aussi d'un fichier local choisi par l'utilisateur
- ② On demande à l'utilisateur ce qu'il veut savoir sur ce dataset :
  - le minimum, le maximum, valeur moyenne sur une plage donnée
  - le nombre de jour où la consommation est supérieure à X (input)

# Description basique du projet

## Et le projet c'est ...

Une analyse et modélisation dynamique de la consommation électrique et de gaz des français !

**Dataset utilisé** (OpenData officiel et mis à jour régulièrement)

<https://odre.opendatasoft.com/explore/dataset/consummation-quotidienne-brute>

## Euh c'est à dire

- ① On laisse la possibilité de récupérer dynamiquement depuis internet mais aussi d'un fichier local choisi par l'utilisateur
- ② On demande à l'utilisateur ce qu'il veut savoir sur ce dataset :
  - le minimum, le maximum, valeur moyenne sur une plage donnée
  - le nombre de jour où la consommation est supérieure à X (input)
  - le nombre de jour où la consommation est inférieure à Y (input)

# Description basique du projet

## Et le projet c'est ...

Une analyse et modélisation dynamique de la consommation électrique et de gaz des français !

**Dataset utilisé** (OpenData officiel et mis à jour régulièrement)

<https://odre.opendatasoft.com/explore/dataset/consummation-quotidienne-brute>

## Euh c'est à dire

- ① On laisse la possibilité de récupérer dynamiquement depuis internet mais aussi d'un fichier local choisi par l'utilisateur
- ② On demande à l'utilisateur ce qu'il veut savoir sur ce dataset :
  - le minimum, le maximum, valeur moyenne sur une plage donnée
  - le nombre de jour où la consommation est supérieure à X (input)
  - le nombre de jour où la consommation est inférieure à Y (input)
- ③ Une représentation graphique de ces données sur laquelle l'utilisateur peut tenter de mettre une fonction pour faire une approximation.

# Plan

- 1 Planning [Rappels]
- 2 GIT : système de gestion de version
  - Présentation rapide
  - HowTo GIT (Local)
  - Mini TP GIT
  - Autre manière de travailler avec GIT
- 3 Démarrage du projet
  - Projet
  - Évaluation

# Évaluation et rendu du travail

## Critères d'évaluation

- Respect du cahier des charges technique et scientifique (10 points)

# Évaluation et rendu du travail

## Critères d'évaluation

- Respect du cahier des charges technique et scientifique (10 points)
- Qualité et originalité de l'interface graphique / code (2.5 points)

# Évaluation et rendu du travail

## Critères d'évaluation

- Respect du cahier des charges technique et scientifique (10 points)
- Qualité et originalité de l'interface graphique / code (2.5 points)
- Aspect commenté du code (5 points)

# Évaluation et rendu du travail

## Critères d'évaluation

- Respect du cahier des charges technique et scientifique (10 points)
- Qualité et originalité de l'interface graphique / code (2.5 points)
- Aspect commenté du code (5 points)
- Intégration sous GIT (2.5 points)



# Évaluation et rendu du travail

## Critères d'évaluation

- Respect du cahier des charges technique et scientifique (10 points)
- Qualité et originalité de l'interface graphique / code (2.5 points)
- Aspect commenté du code (5 points)
- Intégration sous GIT (2.5 points)

# Évaluation et rendu du travail

## Critères d'évaluation

- Respect du cahier des charges technique et scientifique (10 points)
- Qualité et originalité de l'interface graphique / code (2.5 points)
- Aspect commenté du code (5 points)
- Intégration sous GIT (2.5 points)

## Bonus :

- Diversité des outils de traitement numériques

# Évaluation et rendu du travail

## Critères d'évaluation

- Respect du cahier des charges technique et scientifique (10 points)
- Qualité et originalité de l'interface graphique / code (2.5 points)
- Aspect commenté du code (5 points)
- Intégration sous GIT (2.5 points)

## Bonus :

- Diversité des outils de traitement numériques
- Permettre à l'utilisateur de définir plus de paramètres utiles.

# Évaluation et rendu du travail

## Critères d'évaluation

- Respect du cahier des charges technique et scientifique (10 points)
- Qualité et originalité de l'interface graphique / code (2.5 points)
- Aspect commenté du code (5 points)
- Intégration sous GIT (2.5 points)

## Bonus :

- Diversité des outils de traitement numériques
- Permettre à l'utilisateur de définir plus de paramètres utiles.
- Le travail fait maison

# Cahier des charges détaillé

I/II

Fond

- 1 langage python uniquement,

# Cahier des charges détaillé

I/II

Fond

- 1 langage python uniquement,
- 2 programmation orientée objet (classes, méthodes, ...),

# Cahier des charges détaillé

I/II

Fond

- 1 langage python uniquement,
- 2 programmation orientée objet (classes, méthodes, ...),
- 3 programmation conforme PEP 8,

# Cahier des charges détaillé

I/II

Fond

- ① langage python uniquement,
- ② programmation orientée objet (classes, méthodes, ...),
- ③ programmation conforme PEP 8,
- ④ commentaires compatibles outils exports (PEP 257 mini / sphinx)



# Cahier des charges détaillé

I/II

Fond

- 1 langage python uniquement,
- 2 programmation orientée objet (classes, méthodes, ...),
- 3 programmation conforme PEP 8,
- 4 commentaires compatibles outils exports (PEP 257 mini / sphinx)
- 5 interface graphique en PyQt / QtDesigner (avec fichier .ui pour séparer le fond de la forme)

# Cahier des charges détaillé

I/II

Fond

- ① langage python uniquement,
- ② programmation orientée objet (classes, méthodes, ...),
- ③ programmation conforme PEP 8,
- ④ commentaires compatibles outils exports (PEP 257 mini / sphinx)
- ⑤ interface graphique en PyQt / QtDesigner (avec fichier .ui pour séparer le fond de la forme)
- ⑥ bibliothèque capable de gérer les fichiers en multiplateforme (je lancerais à minima sous mon Linux et sous un Windows)

# Cahier des charges détaillé

I/II

Fond

- 1 langage python uniquement,
- 2 programmation orientée objet (classes, méthodes, ...),
- 3 programmation conforme PEP 8,
- 4 commentaires compatibles outils exports (PEP 257 mini / sphinx)
- 5 interface graphique en PyQt / QtDesigner (avec fichier .ui pour séparer le fond de la forme)
- 6 bibliothèque capable de gérer les fichiers en multiplateforme (je lancerais à minima sous mon Linux et sous un Windows)
- 7 code propre et « minimal »

# Cahier des charges détaillé

II/II

## Forme

- ❶ L'interface graphique doit être *conviviale*, **simple**, polyvalente et fonctionnelle

# Cahier des charges détaillé

II/II

## Forme

- ❶ L'interface graphique doit être *conviviale*, **simple**, polyvalente et fonctionnelle
- ❷ La fenêtre graphique doit comporter [à minima] :

# Cahier des charges détaillé

II/II

## Forme

- ① L'interface graphique doit être *conviviale*, **simple**, polyvalente et fonctionnelle
- ② La fenêtre graphique doit comporter [à minima] :
  - Un menu avec des raccourcis claviers

# Cahier des charges détaillé

II/II

## Forme

- ❶ L'interface graphique doit être *conviviale*, **simple**, polyvalente et fonctionnelle
- ❷ La fenêtre graphique doit comporter [à minima] :
  - Un menu avec des raccourcis claviers
  - Des widgets centraux dont un pour chercher le(s) fichier(s)

# Cahier des charges détaillé

II/II

## Forme

- ① L'interface graphique doit être *conviviale*, **simple**, polyvalente et fonctionnelle
- ② La fenêtre graphique doit comporter [à minima] :
  - Un menu avec des raccourcis claviers
  - Des widgets centraux dont un pour chercher le(s) fichier(s)
  - Une barre des tâches listant les événements majeurs de ce qui se passe



# Cahier des charges détaillé

II/II

## Forme

- ① L'interface graphique doit être *conviviale*, **simple**, polyvalente et fonctionnelle
- ② La fenêtre graphique doit comporter [à minima] :
  - Un menu avec des raccourcis claviers
  - Des widgets centraux dont un pour chercher le(s) fichier(s)
  - Une barre des tâches listant les événements majeurs de ce qui se passe
- ③ Le programme doit donner, en ligne de commande toutes les informations clefs des traitements

# Cahier des charges détaillé

II/II

## Forme

- ❶ L'interface graphique doit être *conviviale*, **simple**, polyvalente et fonctionnelle
- ❷ La fenêtre graphique doit comporter [à minima] :
  - Un menu avec des raccourcis claviers
  - Des widgets centraux dont un pour chercher le(s) fichier(s)
  - Une barre des tâches listant les événements majeurs de ce qui se passe
- ❸ Le programme doit donner, en ligne de commande toutes les informations clefs des traitements
- ❹ Votre logiciel doit utiliser les fonctionnalités / méthodes d'au moins deux bibliothèques autres que Python et PyQt

# Cahier des charges détaillé

II/II

## Forme

- ❶ L'interface graphique doit être *conviviale*, **simple**, polyvalente et fonctionnelle
- ❷ La fenêtre graphique doit comporter [à minima] :
  - Un menu avec des raccourcis claviers
  - Des widgets centraux dont un pour chercher le(s) fichier(s)
  - Une barre des tâches listant les événements majeurs de ce qui se passe
- ❸ Le programme doit donner, en ligne de commande toutes les informations clefs des traitements
- ❹ Votre logiciel doit utiliser les fonctionnalités / méthodes d'au moins deux bibliothèques autres que Python et PyQt
  - dont une pour la gestion des fichiers (textes, sons, csv, etc)

# Cahier des charges détaillé

II/II

## Forme

- ❶ L'interface graphique doit être *conviviale*, **simple**, polyvalente et fonctionnelle
- ❷ La fenêtre graphique doit comporter [à minima] :
  - Un menu avec des raccourcis claviers
  - Des widgets centraux dont un pour chercher le(s) fichier(s)
  - Une barre des tâches listant les événements majeurs de ce qui se passe
- ❸ Le programme doit donner, en ligne de commande toutes les informations clefs des traitements
- ❹ Votre logiciel doit utiliser les fonctionnalités / méthodes d'au moins deux bibliothèques autres que Python et PyQt
  - dont une pour la gestion des fichiers (textes, sons, csv, etc)
  - et au moins une pour les mathématiques impliqués dans les traitements.

# Licence de ce document

# Merci de votre attention !

## Licence du document

Ce travail est sous licence GFDL  
(ou licence *GNU Free Documentation Licence.*)

<http://www.gnu.org/licenses/fdl.html>

Notez que cette présentation ne possède

- pas de section inaltérable
- pas de texte de première page de couverture
- pas de texte de dernière page de couverture