

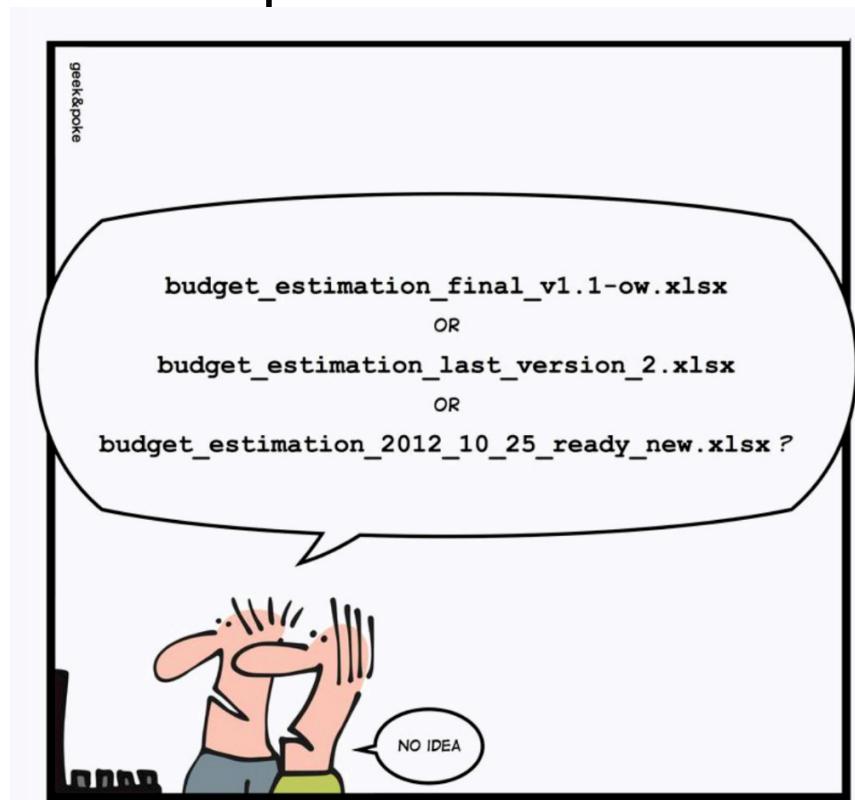
Informatique II

Gestion de versions



Comment gérer un projet ?

- Pour un projet **seul** :
 - Comment gérer les différentes versions/ les historiques du projet.
 - Se souvenir de pourquoi des modifications ont été faites.
 - Quels fichiers font partie de la version stable ? Sont des tests ?



VERSION CONTROL

from Geek&Poke

Comment gérer un projet ?

- Pour un projet **en équipe** :
 - Même problématiques que quand on code tout seul
 - Comment partager ses avancées avec les autres? Où stocker les fichiers du projet?
 - Comment modifier un code sans perturber le travail des autres ?
 - Avoir une référence commune pour le projet.
 - Savoir qui a fait quoi.



La gestion de configuration

- La **gestion de configuration** consiste à :
 - gérer la description technique d'un système
 - gérer l'ensemble des modifications apportées au cours de l'évolution de ce système.
- Tout impact sur l'application doit être tracé. On peut rechercher et comparer les différentes version d'une application.
- Des méthodes doivent être appliquées pour garantir ce niveau de contrainte, mais ce n'est pas suffisant.
- Des **outils** doivent être utilisés pour garantir fiabilité / intégrité, et ainsi d'éviter au maximum les opérations humaines

Les gestionnaires de versions

- Il existe des outils appelés **gestionnaires de versions**, ou **outils de versionnage** qui permettent de faire de la gestion de configurations en partie automatique.
- L'utilisation de ces outils représente l'une des obligations du développeur quand il travaille sur une application.
- Ces outils permettent de tracer toute modification dans le code, même mineure.



Le but des outils de versionnage

- Avoir un espace centralisé sur lequel le code de l'application est stocké :
 - Garantir une référence unique
 - Accessible à tout le monde /ou membre de l'équipe
- Gérer et sauvegarder les différentes versions (et avancées) d'un projet:
 - Avoir un historique des modifications
 - Pouvoir revenir à une version précédente
 - Faire une dichotomie dans l'historique des recherches pour retrouver l'origine d'un bug
 - Garantir que l'on est capable de regénérer exactement le même exécutable
 - Apporter des solutions avec des fils de discussions, des documents ect.

Les outils de versionnage

- Les outils de gestion de versions pourraient faire l'objet d'un module de cours à eux seuls
- Ce cours n'a pour seul but que de vous sensibiliser à l'existence de ce type d'outils, vous informer sur la manière dont ils fonctionnent, et vous inciter à les utiliser dans vos projets académiques et/ou personnels.
- La question d'utiliser un tel type d'outils dans un contexte professionnel ne se pose pas : c'est une obligation (une de plus) pour le développeur.
- Initialiser un dépôt de code (terme consacré) pour archiver les futures modifications d'une application devrait être un réflexe, avant même de concevoir ou coder (et c'est un conseil qui vaut aussi pour vos projets académiques/persos)

Les outils de versionnage

- Attention, il y a de mauvais côtés à utiliser ce type d'outils



Git



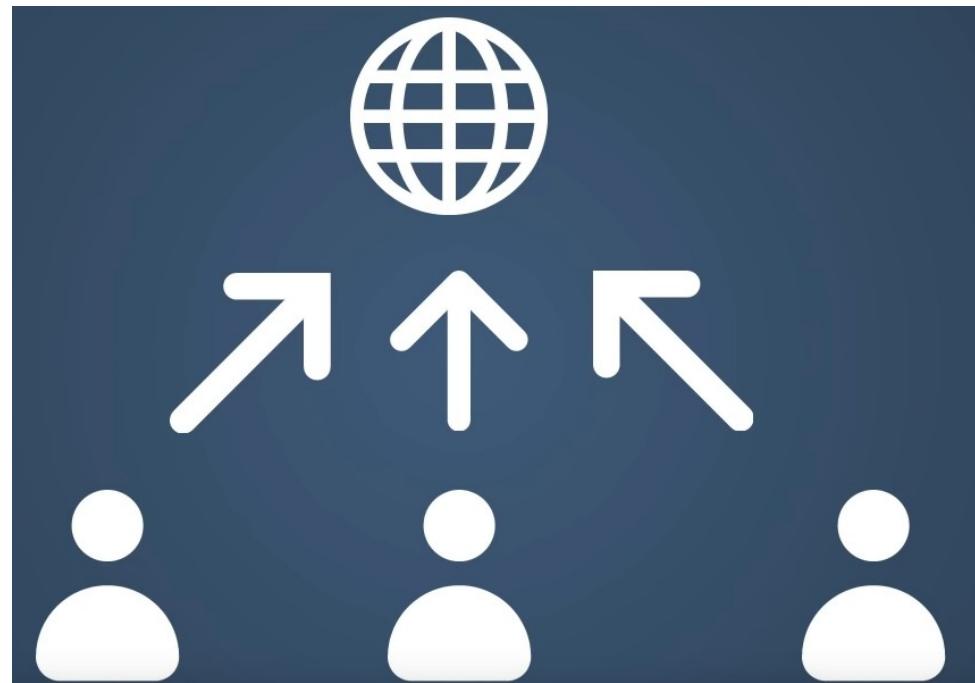
Le logiciel Git : petit historique

- Linux est un logiciel libre : un outil de versionning est tout indispensable pour gérer les **très** nombreux apports et version de l'OS.
- Le logiciel BitKeeper était utilisé jusqu'en 2002, mais est devenu payant.
- En 2005, Linus Torvalds propose GIT qui devient rapidement un des outils de versioning les plus utilisé.



Le logiciel Git

- La principale caractéristique de GIT est qu'il ne repose pas sur un serveur centralisé unique.



- Gestionnaire avec serveur centralisé : tous les utilisateurs partagent un même ensemble de fichiers stockés sur un serveur

Le logiciel Git

- La principale caractéristique de GIT est qu'il ne repose pas sur un serveur centralisé unique.



- Gestionnaire avec serveur centralisé ET copie locale : chaque utilisateur peut contribuer au dépôt principal en synchronisant le dépôt local sur sa machine

Le logiciel Git

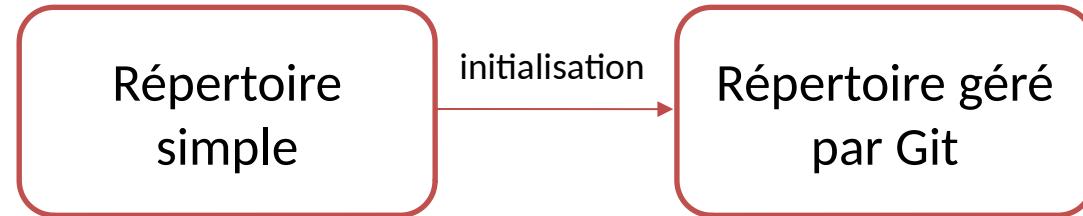
- La principale caractéristique de GIT est qu'il ne repose pas sur un serveur centralisé unique.
- Il existe plusieurs solutions permettant d'héberger le dépôt central d'un projet géré avec Git
- L'un d'eux est GitHub.com (il faudra vous créer un compte!)



La mascotte s'appelle octocat !

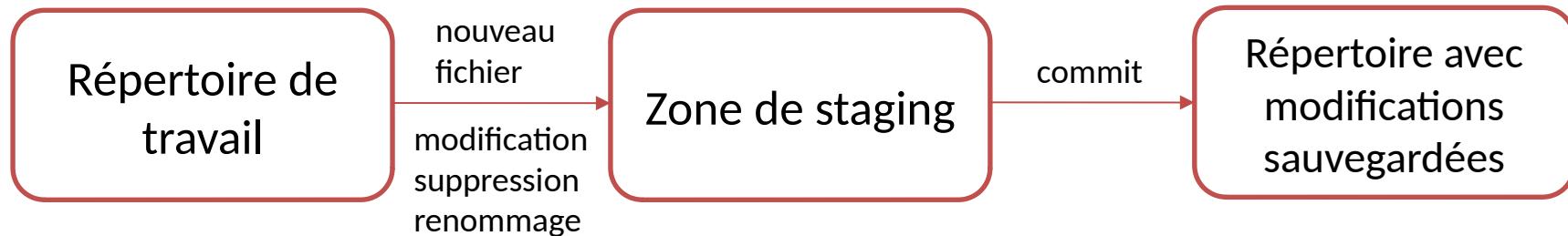
Le versioning en local avec Git

- Au tout départ, il faut **initialiser** un espace de travail qui sera géré par Git
- On peut partir d'un nouveau répertoire vide, ou déjà existant, en indiquant à Git que ce répertoire deviendra la racine de notre projet
- L'exécutable Git possède une commande d'initialisation qui va créer un dossier caché dans le répertoire de travail : '**.git**'
- Ce dossier **.git** contiendra l'ensemble de l'historique des modifications ultérieures



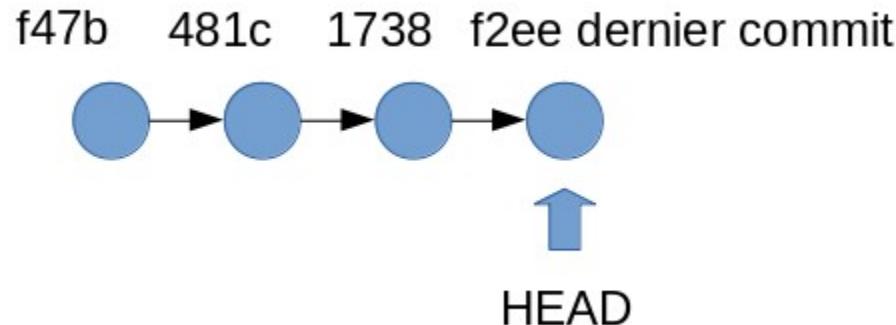
Le versioning en local avec Git

- Quand un développeur effectue une modification de son code et qu'il souhaite soumettre cette modification, on parle de **commit**.
- Il peut choisir les changements qu'il soumet dans le commit en ajoutant des fichiers à la **zone de staging**
- Les différents commits sont stockés dans un espace mémoire **persistant** : le dossier `.git` vu précédemment
- Les commits sont donc des opérations issues de décisions humaines : pas de sauvegarde automatique par Git



Le versioning en local avec Git

- En général chaque différence de fichier est stockée plutôt que l'intégralité du fichier lui-même
- Chaque commit est donc **signé** de manière unique permettant de s'assurer de l'intégrité des données : on parle de **version** de commit, ou de **révision**
- Un lien chronologique est créé entre les différentes révisions, ce qui permet d'obtenir un graphe de révisions



Le versioning en local avec Git

- Une fois les commits effectués, on peut obtenir des informations sur l'historique
- La date, l'auteur et le message font partie de ces informations. Le message devant inclure le maximum de précisions possibles concernant le contenu des modifications
- Cet historique peut être consulté globalement, ou fichier par fichier, pour permettre de retrouver la cause d'un bug, ou tout simplement voir le détail d'une modification

commit **1144223e4d741d4c6ad2d34278d10a1c8404518a**

Author: Romuald GRIGNON <rgn@cy-tech.fr>

Date: Fri Apr 15 18:57:40 2022 +0200

First commit for test

...

...

Démos :

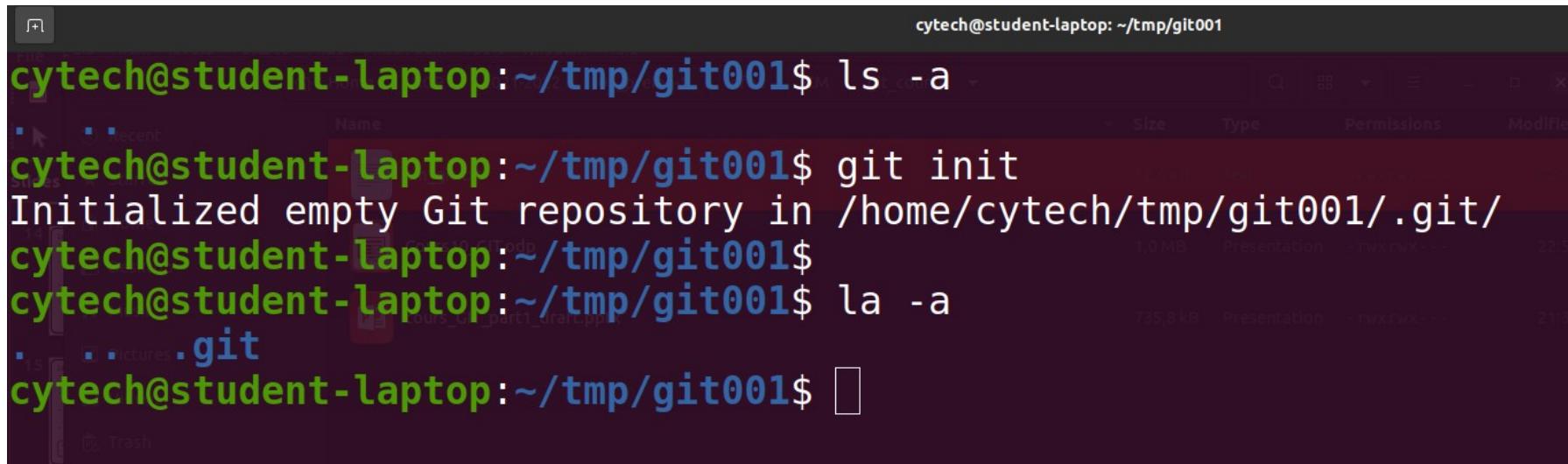
- Les sections qui suivent, intitulées “Démo”, serviront à vous présenter de manière interactive les différentes commandes Git et les situations dans lesquelles les employer
- Cela pourra vous servir de tutoriel si vous souhaitez refaire les manipulations chez vous plus tard
- Elles sont séparées en plusieurs parties :
 - Démo 1 : dépôt local pour un seul développeur
 - Démo 2 : dépôt local avec annulation de modifications
 - Démo 3 : liaison du dépôt local avec un dépôt central (pour un seul développeur)
 - Démo 4 : dépôt central avec gestion des conflits
 - Démo 5 : les branches

Démo 1 : dépôt local



Démo 1 : dépôt local

- Créer un dossier vide
- Se placer dans le dossier et taper la commande **git init**



The screenshot shows a terminal window with a dark background. The title bar indicates the user is on a student laptop at the home directory of /tmp/git001. The terminal output is as follows:

```
cytech@student-laptop:~/tmp/git001$ ls -a
.
..
cytech@student-laptop:~/tmp/git001$ git init
Initialized empty Git repository in /home/cytech/tmp/git001/.git/
cytech@student-laptop:~/tmp/git001$ 
cytech@student-laptop:~/tmp/git001$ ls -a
.
..
.git
cytech@student-laptop:~/tmp/git001$ 
```

The terminal shows the creation of a new Git repository in the current directory. A hidden directory named '.git' is created, which is only visible when using the '-a' option with the 'ls' command.

- Un sous dossier `.git` a été créé
- C'est un dossier caché, il n'apparaît qu'avec l'option `-a` de la commande `ls`

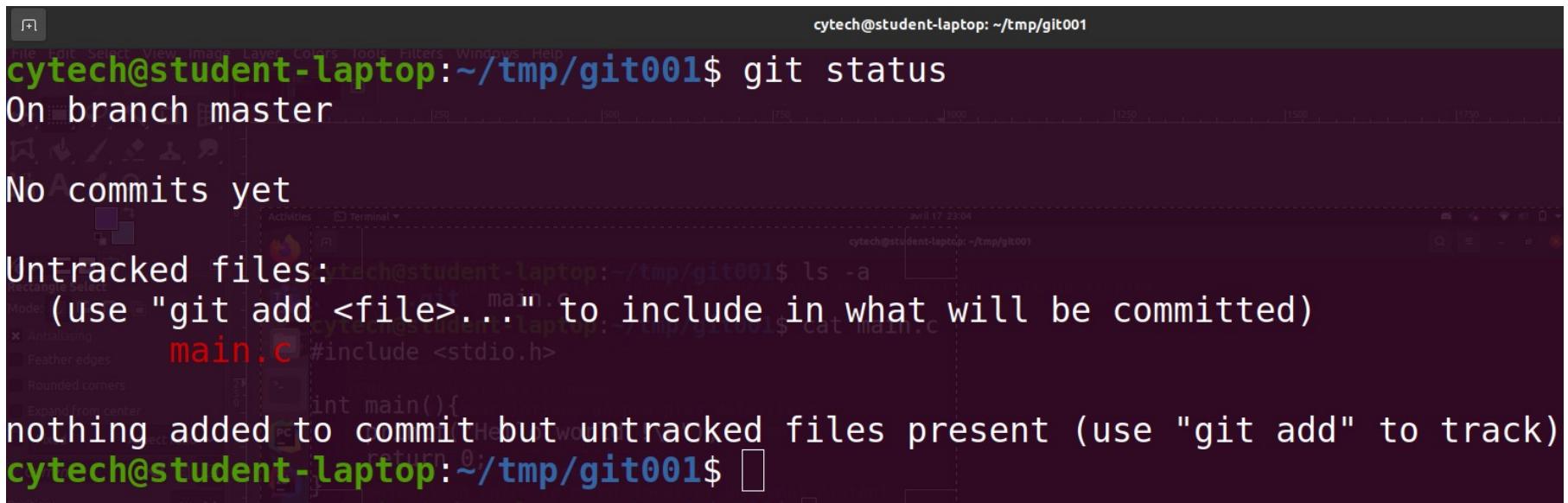
Démo 1 : dépôt local

- Créer un fichier texte et ajouter du contenu (ici un fichier main.c)

```
cytech@student-laptop:~/tmp/git001$ ls -a
.. .git main.c
cytech@student-laptop:~/tmp/git001$ cat main.c
#include <stdio.h>
int main(){
    printf("Hello world !\n");
    return 0;
}
cytech@student-laptop:~/tmp/git001$ 
```

Démo 1 : dépôt local

- Taper la commande ***git status*** qui permet de voir l'état du dépôt



The screenshot shows a terminal window on a Linux desktop. The terminal output is as follows:

```
cytech@student-laptop:~/tmp/git001$ git status
On branch master
No commits yet
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    main.c #include <stdio.h>
nothing added to commit but untracked files present (use "git add" to track)
cytech@student-laptop:~/tmp/git001$
```

The terminal window has a dark background with light-colored text. The desktop environment includes a dock with icons for various applications like a browser, file manager, and terminal, and a system tray at the bottom.

- On voit que git a détecté le fichier main.c comme étant “untracked”, c'est à dire que git ne le connaît pas
- Ce fichier ne fait donc pas (encore) partie des fichiers dont il doit suivre les modifications

Démo 1 : dépôt local

- Ajouter le fichier main.c à la zone de staging grâce à la commande **git add ...** (visualiser le résultat avec **git status**)

```
cytech@student-laptop:~/tmp/git001$ git add main.c
cytech@student-laptop:~/tmp/git001$ git status
On branch master
No commits yet
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   main.c

cytech@student-laptop:~/tmp/git001$
```

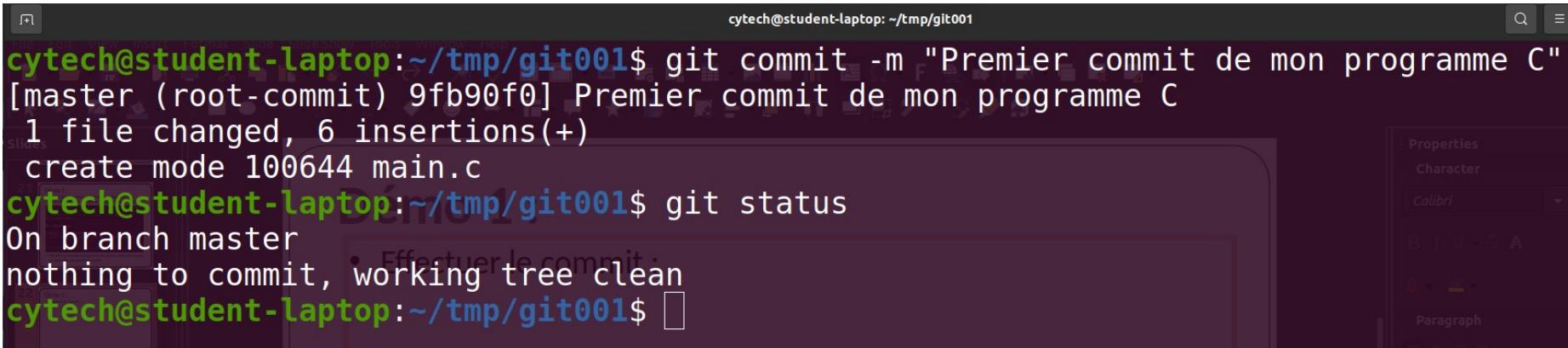
The screenshot shows a terminal window with the following content:

- Terminal title: cytech@student-laptop: ~/tmp/git001\$
- Command: git add main.c
- Command: git status
- Output:
 - No commits yet
 - Changes to be committed:
 - (use "git rm --cached <file>..." to unstage)
 - new file: main.c
- Final command: cytech@student-laptop:~/tmp/git001\$

- Git détecte maintenant que le fichier main.c doit être soumis au dépôt lors du prochain commit

Démo 1 : dépôt local

- Effectuer le commit à l'aide de la commande **git commit ...** :



The screenshot shows a terminal window with the following content:

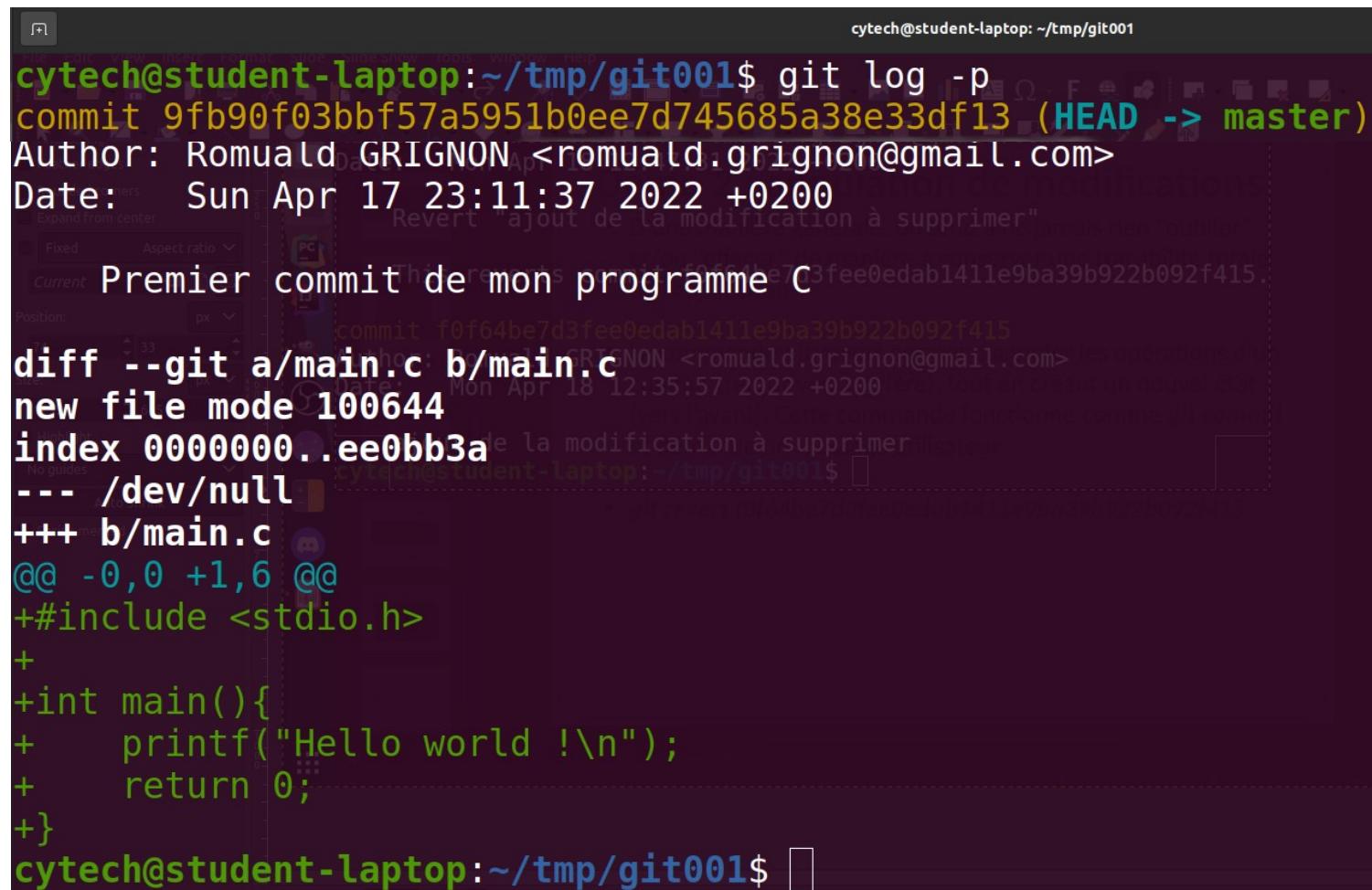
```
cytech@student-laptop:~/tmp/git001$ git commit -m "Premier commit de mon programme C"
[master (root-commit) 9fb90f0] Premier commit de mon programme C
 1 file changed, 6 insertions(+)
  create mode 100644 main.c
cytech@student-laptop:~/tmp/git001$ git status
On branch master
nothing to commit, working tree clean
cytech@student-laptop:~/tmp/git001$
```

A red box highlights the terminal window, and a red arrow points from the text "Effectuer le commit" to the terminal window.

- On s'aperçoit que le commit a fonctionné, et que le dépôt local est à jour (aucune modification : working tree clean)
- L'option **-m** de la commande **git commit**, permet d'entrer le message du commit. L'omettre va ouvrir l'éditeur de texte du système par défaut (sous Linux cela sera probablement “vi”)
- Pour changer l'éditeur par défaut, par exemple gedit :
 - **git config --global core.editor gedit**

Démo 1 : dépôt local

- Pour visualiser le commit précédent, utilisez la commande suivante : **git log -p**
- L'option **-p** vous donne les détails de la modification du fichier



```
cytech@student-laptop:~/tmp/git001$ git log -p
commit 9fb90f03bbf57a5951b0ee7d745685a38e33df13 (HEAD -> master)
Author: Romuald GRIGNON <romuald.grignon@gmail.com>
Date:   Sun Apr 17 23:11:37 2022 +0200

        Premier commit de mon programme C

diff --git a/main.c b/main.c
new file mode 100644
index 000000..ee0bb3a
--- /dev/null
+++ b/main.c
@@ -0,0 +1,6 @@
+#include <stdio.h>
+
+int main(){
+    printf("Hello world !\n");
+    return 0;
+}
cytech@student-laptop:~/tmp/git001$
```

Démo 1 : dépôt local

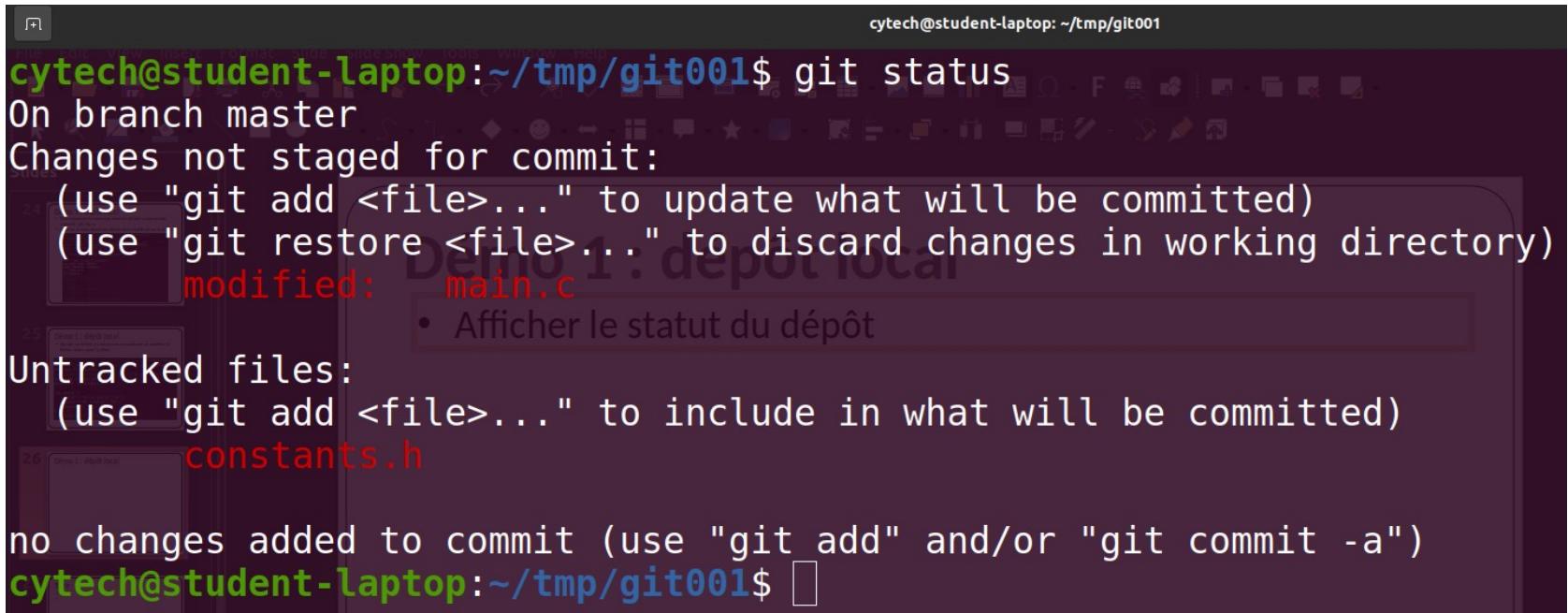
- Ajouter un fichier .h contenant une constante et modifier le fichier main.c pour l'utiliser

```
cytech@student-laptop:~/tmp/git001$ cat constants.h
#define PI 3.14159
cytech@student-laptop:~/tmp/git001$ cat main.c
#include <stdio.h>
#include "constants.h"

int main(){
    printf("Hello world!\n");
    printf("PI = %f\n", PI);
    return 0;
}
cytech@student-laptop:~/tmp/git001$
```

Démo 1 : dépôt local

- Afficher le statut du dépôt



The screenshot shows a terminal window with the following text:

```
cytech@student-laptop:~/tmp/git001$ 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:   main.c

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    constants.h

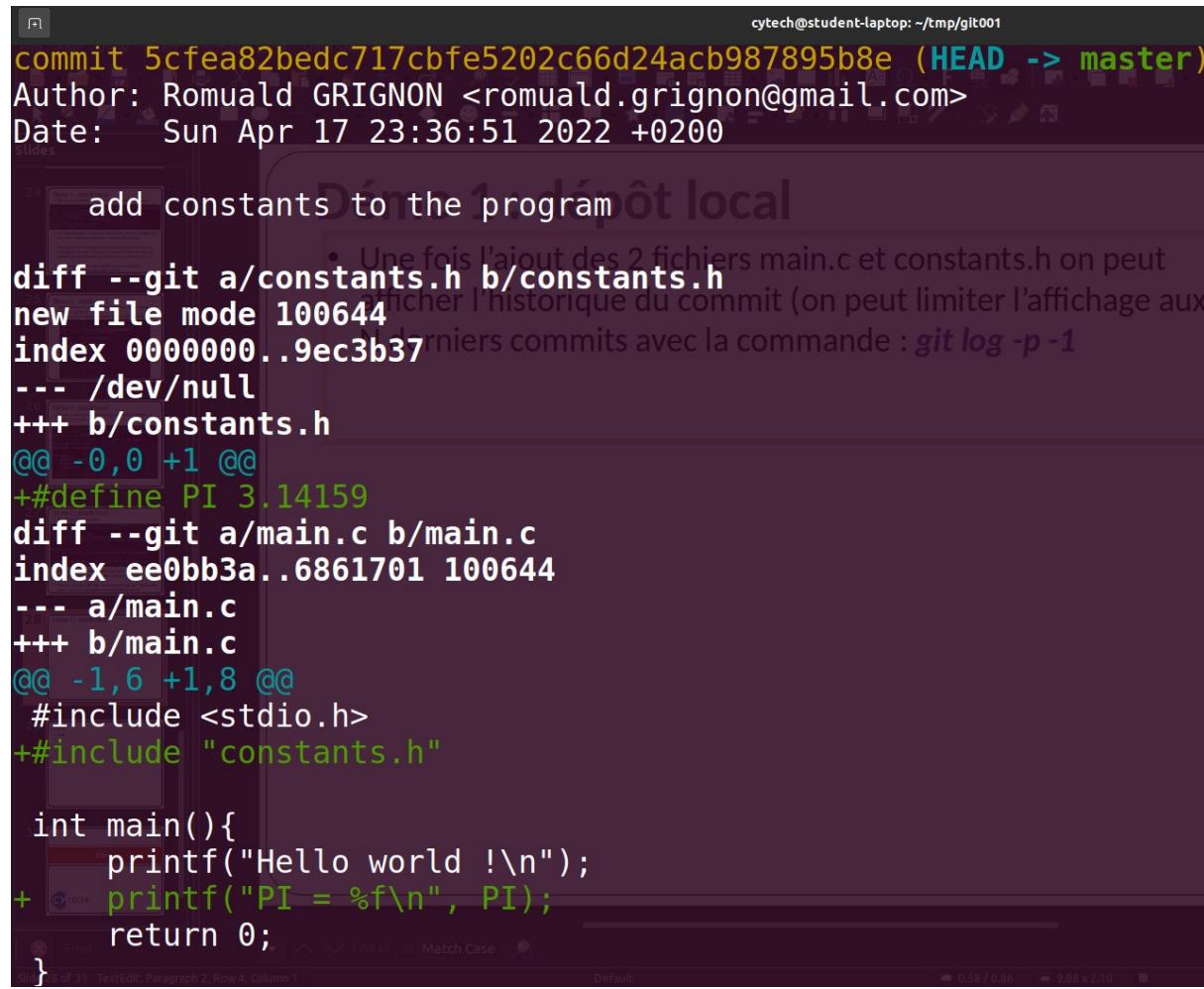
no changes added to commit (use "git add" and/or "git commit -a")
cytech@student-laptop:~/tmp/git001$
```

A blue double arrow icon is positioned to the left of the terminal window. A callout bubble with a blue border and white text is overlaid on the terminal window, containing the text: "• Afficher le statut du dépôt".

- Nous voyons apparaître un nouveau fichier non connu de git (constants.h) et un fichier modifié (main.c)
- Là aussi, il faudra effectuer les commandes git add / git commit pour avoir une sauvegarde des modifications par Git

Démo 1 : dépôt local

- Une fois l'ajout des 2 fichiers main.c et constants.h on peut afficher l'historique du commit (on peut limiter l'affichage au dernier commit avec la commande : **git log -p -1**)



```
cytech@student-laptop: ~/tmp/git001
commit 5cfea82bedc717cbfe5202c66d24acb987895b8e (HEAD -> master)
Author: Romuald GRIGNON <romuald.grignon@gmail.com>
Date:   Sun Apr 17 23:36:51 2022 +0200

    add constants to the program

diff --git a/constants.h b/constants.h
new file mode 100644
index 0000000..9ec3b37
--- /dev/null
+++ b/constants.h
@@ -0,0 +1 @@
+#define PI 3.14159
diff --git a/main.c b/main.c
index ee0bb3a..6861701 100644
--- a/main.c
+++ b/main.c
@@ -1,6 +1,8 @@
#include <stdio.h>
+#include "constants.h"

int main(){
    printf("Hello world !\n");
+    printf("PI = %f\n", PI);
    return 0;
}
```

Démo 2 : annulation de modifications



Démo 2 : annulation de modifications

- Nous allons effectuer une modification sur le fichier main.c puis nous allons revenir sur notre choix



```
cytech@student-laptop:~/tmp/git001$ 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:   main.c
no changes added to commit (use "git add" and/or "git commit -a")
cytech@student-laptop:~/tmp/git001$ 
```

- La commande **git restore** vous permet d'annuler les modifications locales et de restaurer l'état du fichier lors du dernier commit

Démo 2 : annulation de modifications

- Nous allons maintenant effectuer une modification sur le fichier main.c puis le passer dans l'état "staged" avant de revenir sur notre choix de le modifier



The screenshot shows a terminal window with the following output:

```
cytech@student-laptop:~/tmp/git001$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   main.c
```

A blue double arrow icon is positioned to the left of the terminal window.

- La commande **git restore --staged** vous permet d'annuler l'état "staged" mais pas les modifications locales
- Il faudra en plus effectuer la commande **git restore**, comme précédemment, pour revenir à l'état d'origine du fichier (si c'est ce que l'on souhaite)

Démo 2 : annulation de modifications

- Nous allons maintenant effectuer une modification sur le fichier main.c puis la sauvegarder (commit)



```
cytech@student-laptop:~/tmp/git001$ git log -3
commit f0f64be7d3fee0edab1411e9ba39b922b092f415 (HEAD -> master)
Author: Romuald GRIGNON <romuald.grignon@gmail.com>
Date: Mon Apr 18 12:35:57 2022 +0200

    ajout de la modification à supprimer

commit 5cfea82bedc717cbfe5202c66d24acb987895b8e
Author: Romuald GRIGNON <romuald.grignon@gmail.com>
Date: Sun Apr 17 23:36:51 2022 +0200

    add constants to the program

commit 9fb90f03bbf57a5951b0ee7d745685a38e33df13
Author: Romuald GRIGNON <romuald.grignon@gmail.com>
Date: Sun Apr 17 23:11:37 2022 +0200

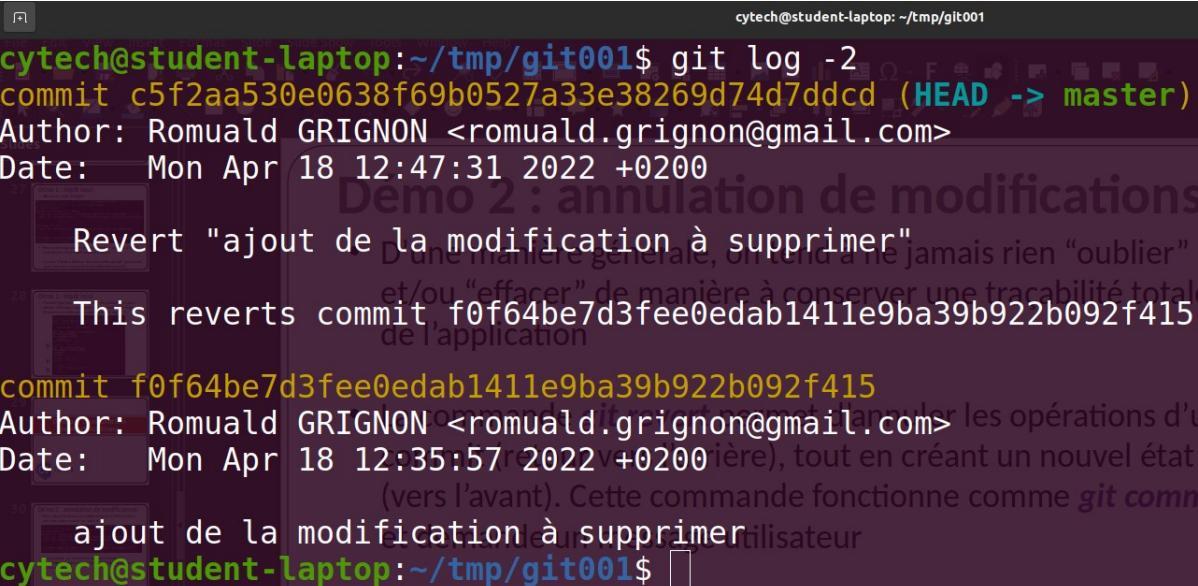
    Premier commit de mon programme C

cytech@student-laptop:~/tmp/git001$
```

- On retrouve dans l'historique notre modification actuelle, précédée de celle où nous avions ajouté le fichier constants.h

Démo 2 : annulation de modifications

- D'une manière générale, on tend à ne jamais rien "oublier" et/ou "effacer" de manière à conserver une traçabilité totale de l'application
- La commande **git revert** permet d'annuler les opérations d'un commit (retour vers l'arrière), tout en créant un nouvel état (vers l'avant). Cette commande fonctionne comme **git commit** et demande un message utilisateur
- **git revert f0f64be7d3fee0edab1411e9ba39b922b092f415**



The screenshot shows a terminal window with the following content:

```
cytech@student-laptop:~/tmp/git001$ git log -2
commit c5f2aa530e0638f69b0527a33e38269d74d7ddcd (HEAD -> master)
Author: Romuald GRIGNON <romuald.grignon@gmail.com>
Date:   Mon Apr 18 12:47:31 2022 +0200

    Revert "ajout de la modification à supprimer"
    This reverts commit f0f64be7d3fee0edab1411e9ba39b922b092f415.

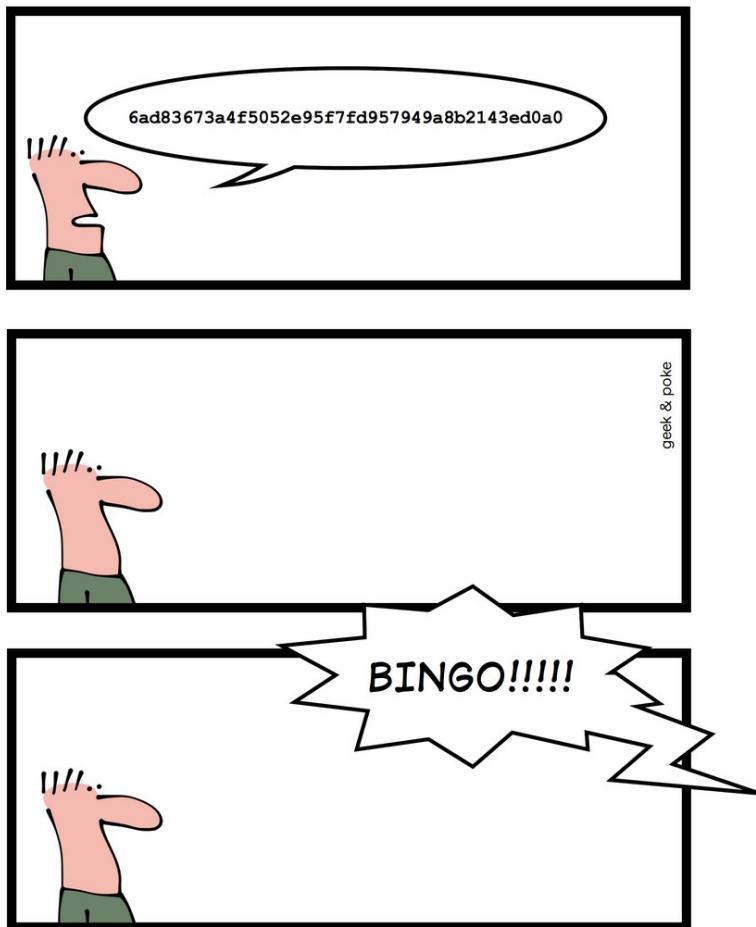
commit f0f64be7d3fee0edab1411e9ba39b922b092f415
Author: Romuald GRIGNON <romuald.grignon@gmail.com>
Date:   Mon Apr 18 12:35:57 2022 +0200

    ajout de la modification à supprimer
cytech@student-laptop:~/tmp/git001$
```

Démo 2 : annulation de modifications

- Comme indiqué précédemment, chaque commit possède une signature unique et c'est celle-ci qui sera utilisée pour y faire référence

Games For The Real Geeks



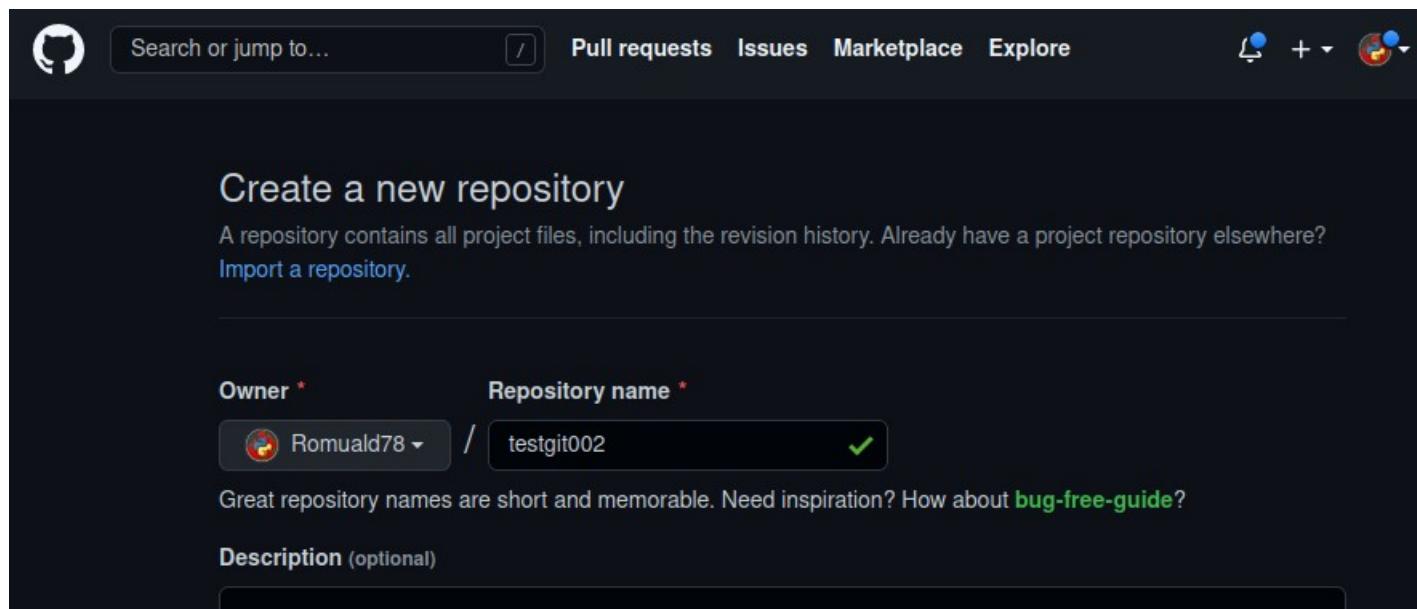
Today: git-bingo

Démo 3 : dépôt central



Démo 3 : dépôt central

- On souhaite créer un dépôt central à partir de notre dépôt local.
- Nous allons utiliser le site github.com en créant un compte puis en créant un nouveau dépôt (vide)



- Il existe d'autres franchises permettant d'héberger un dépôt git gratuitement : GitLab / BitBucket / SourceForge / ...

Démo 3 : dépôt central

- Toutes les modifications de code sont faites dans ce que l'on appelle une **branche**
- Une branche est une variation de l'application : gérer plusieurs branches permet de gérer plusieurs modifications en parallèle sans se perturber les unes avec les autres
- Par défaut notre dépôt local travaille sur la branche “master”. On peut le voir grâce à la commande **git branch**



```
cytech@student-laptop:~/tmp/git001$ git branch -a
* master
cytech@student-laptop:~/tmp/git001$ 
```

Démo 3 : dépôt central

- Par défaut le site de `github.com` crée une branche “*main*”
- Pour relier les 2 dépôts (local et distant) il faut que les branches soient alignées : si elles portent des noms différents, git interprétera ces 2 branches comme distinctes
- On peut forcer le renommage de la branche actuellement sélectionnée avec la commande ***git branch -M*** en donnant le nouveau nom : ici “*main*” au lieu de “*master*”



A screenshot of a terminal window titled "Démo 3 : dépôt central". The terminal shows the following commands being run:

```
cytech@student-laptop: ~/tmp/git001$ git branch -M main
cytech@student-laptop: ~/tmp/git001$ git branch -a
* main
cytech@student-laptop: ~/tmp/git001$
```

The terminal window has a dark background with light-colored text. The prompt is green, and the output of the commands is blue. A small watermark "Démo 3 : dépôt central" is visible in the bottom right corner of the slide.

Démo 3 : dépôt central

- Maintenant nous pouvons relier les 2 dépôts
- c'est la commande ***git remote add*** qui permet d'indiquer où se trouve le dépôt distant
- On donne un nom au dépôt distant : très souvent vous trouverez l'exemple de “***origin***”



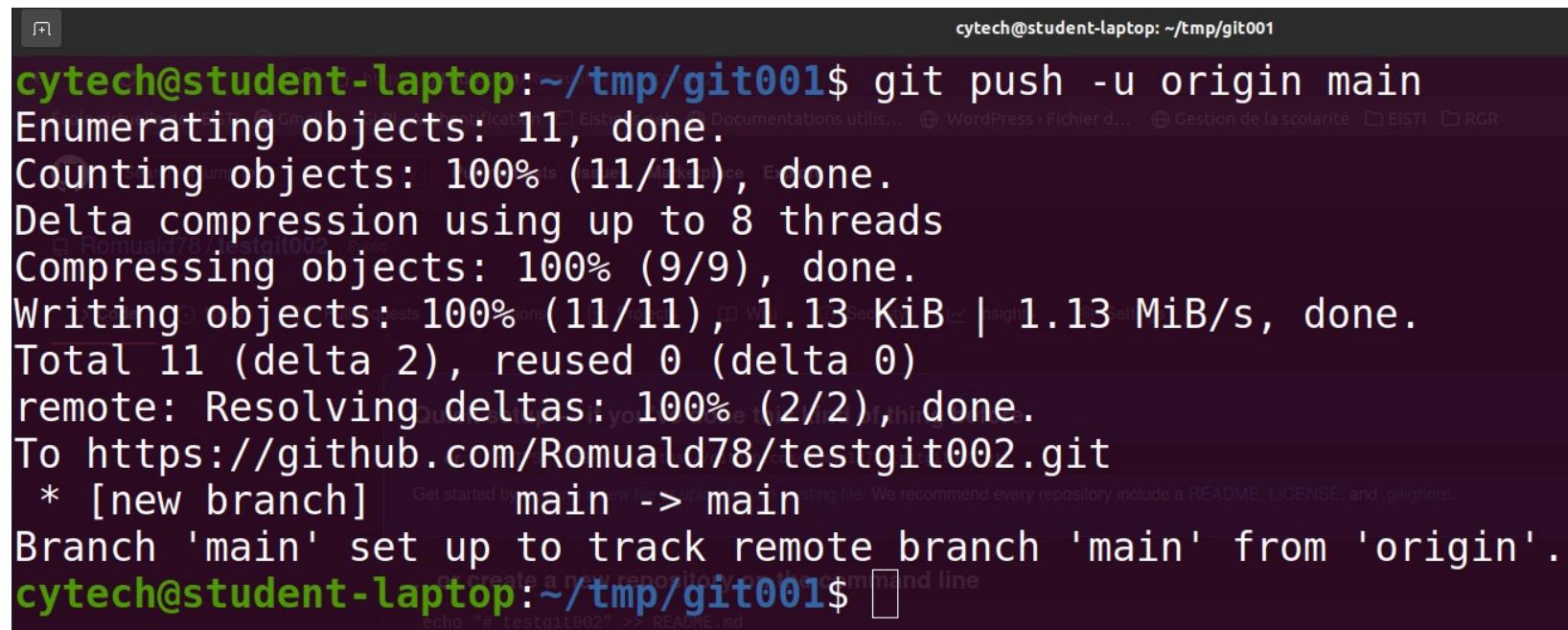
```
cytech@student-laptop:~/tmp/git001$ git remote add origin https://github.com/Romuald78/testgit002.git
cytech@student-laptop:~/tmp/git001$ git remote show
origin
```



```
cytech@student-laptop:~/tmp/git001$ git remote -v
origin  https://github.com/Romuald78/testgit002.git (fetch)
origin  https://github.com/Romuald78/testgit002.git (push)
cytech@student-laptop:~/tmp/git001$
```

Démo 3 : dépôt central

- Maintenant nous pouvons mettre à jour le dépôt distant (qui est vide) avec le contenu de notre dépôt local
- Les synchronisations entre le dépôt local et distant se font à l'aide de 2 commandes “**git push**” et “**git pull**”



```
cytech@student-laptop:~/tmp/git001$ git push -u origin main
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 8 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (11/11), 1.13 KiB | 1.13 MiB/s, done.
Total 11 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/Romuald78/testgit002.git
 * [new branch] main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
cytech@student-laptop:~/tmp/git001$
```

- Attention : il faut être authentifié vis-a-vis du site github.com
- Utiliser un **Personal Access Token** (cf. détails sur github.com)

Démo 3 : dépôt central

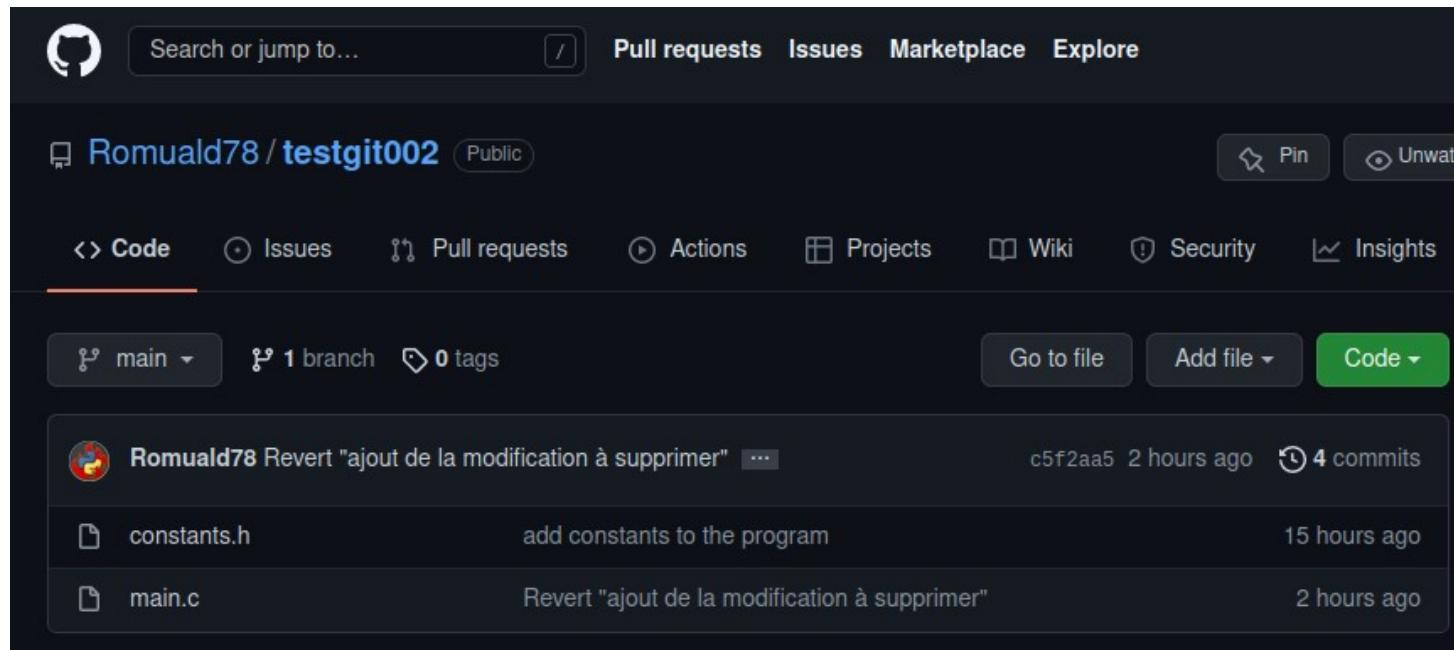
- Pour récupérer les mises à jour depuis le dépôt central, utiliser la commande **git pull**



```
cytech@stud... ~]$ git pull
Already up to date.
cytech@stud... ~]$
```

A screenshot of a terminal window titled "cytech@student". It shows the command "git pull" being run, followed by the message "Already up to date." and a final prompt "cytech@stud... ~]\$". The background of the terminal is dark, and the text is white.

- Bien évidemment notre dépôt est à jour. Sur le site github.com nous retrouvons l'ensemble des fichiers de l'état courant



Romuald78 / testgit002 Public

Code Issues Pull requests Actions Projects Wiki Security Insights

main 1 branch 0 tags

Romuald78 Revert "ajout de la modification à supprimer" ... c5f2aa5 2 hours ago 4 commits

constants.h add constants to the program 15 hours ago

main.c Revert "ajout de la modification à supprimer" 2 hours ago

A screenshot of a GitHub repository page for "Romuald78 / testgit002". The repository is public. The "Code" tab is selected. It shows one branch ("main") and zero tags. Below the navigation bar, there are buttons for "Go to file", "Add file", and "Code". A list of commits is displayed, with the most recent commit being a revert of an addition. The commit details show it was made 2 hours ago and includes four commits. Below the commits, two specific files are listed: "constants.h" and "main.c", each with a brief description and timestamp.

Démo 3 : dépôt central

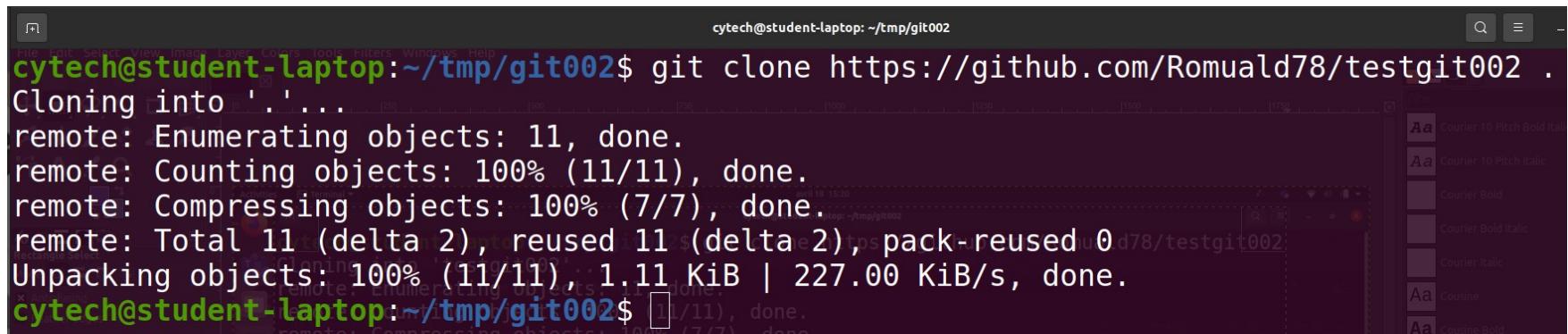
- Maintenant que nous avons un dépôt local et distant, il est possible de continuer nos développements
- La même séquence de commandes sera utilisée :
 - Modifications (ajouts, modifications, suppressions, renommage, déplacements)
 - Choix des modifications à sauver (commande **git add**)
 - Sauvegarde (commande **git commit**)
 - Synchronisation (commande **git push**)
- Une bonne pratique est de soumettre ses modifications quand elles apportent un ajout fonctionnel (même minime)
- Effectuer un commit quotidien au minimum, si les modifications sont longues, ou si le développeur doit s'interrompre (attention à ne pas casser le fonctionnement (cf. §branches ci-après))

Démo 4 : dépôt central et conflits



Démo 4 : ajout d'un collaborateur

- Maintenant nous avons la possibilité de travailler à plusieurs
- Nous allons créer un autre répertoire pour simuler un nouveau collaborateur
- Créer un répertoire vide et se placer à l'intérieur.
- Utiliser la commande **git clone** pour récupérer l'état courant de l'application et faire le lien entre le nouveau dépôt local et le dépôt distant



A screenshot of a terminal window titled "cytech@student-laptop: ~/tmp/git002". The window shows the command "git clone https://github.com/Romuald78/testgit002 ." being run. The output of the command is displayed, showing the progress of cloning the repository, including object enumeration, counting, compressing, and unpacking. The terminal has a dark theme with light-colored text. A vertical toolbar on the right side contains icons for file operations like copy, paste, and search, along with font and color selection tools.

```
cytech@student-laptop:~/tmp/git002$ git clone https://github.com/Romuald78/testgit002 .
Cloning into '.'...
remote: Enumerating objects: 11, done.
remote: Counting objects: 100% (11/11), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 11 (delta 2), reused 11 (delta 2), pack-reused 0
Unpacking objects: 100% (11/11), 1.11 KiB | 227.00 KiB/s, done.
cytech@student-laptop:~/tmp/git002$
```

Démo 4 : ajout d'un collaborateur

- On peut vérifier que l'on possède une branche “main” en local
- Que le dépôt distant est bien connecté à la cible “origin”
- Que le statut du dépôt nouvellement créé est stable

```
cytech@student-laptop:~/tmp/git002$ git branch
* main
cytech@student-laptop:~/tmp/git002$ git remote -v
origin https://github.com/Romuald78/testgit002 (fetch)
origin https://github.com/Romuald78/testgit002 (push)
cytech@student-laptop:~/tmp/git002$ git status
On branch main
    Your branch is up to date with 'origin/main'.

        nothing to commit, working tree clean
cytech@student-laptop:~/tmp/git002$ █
```

Démo 4 : ajout d'un collaborateur

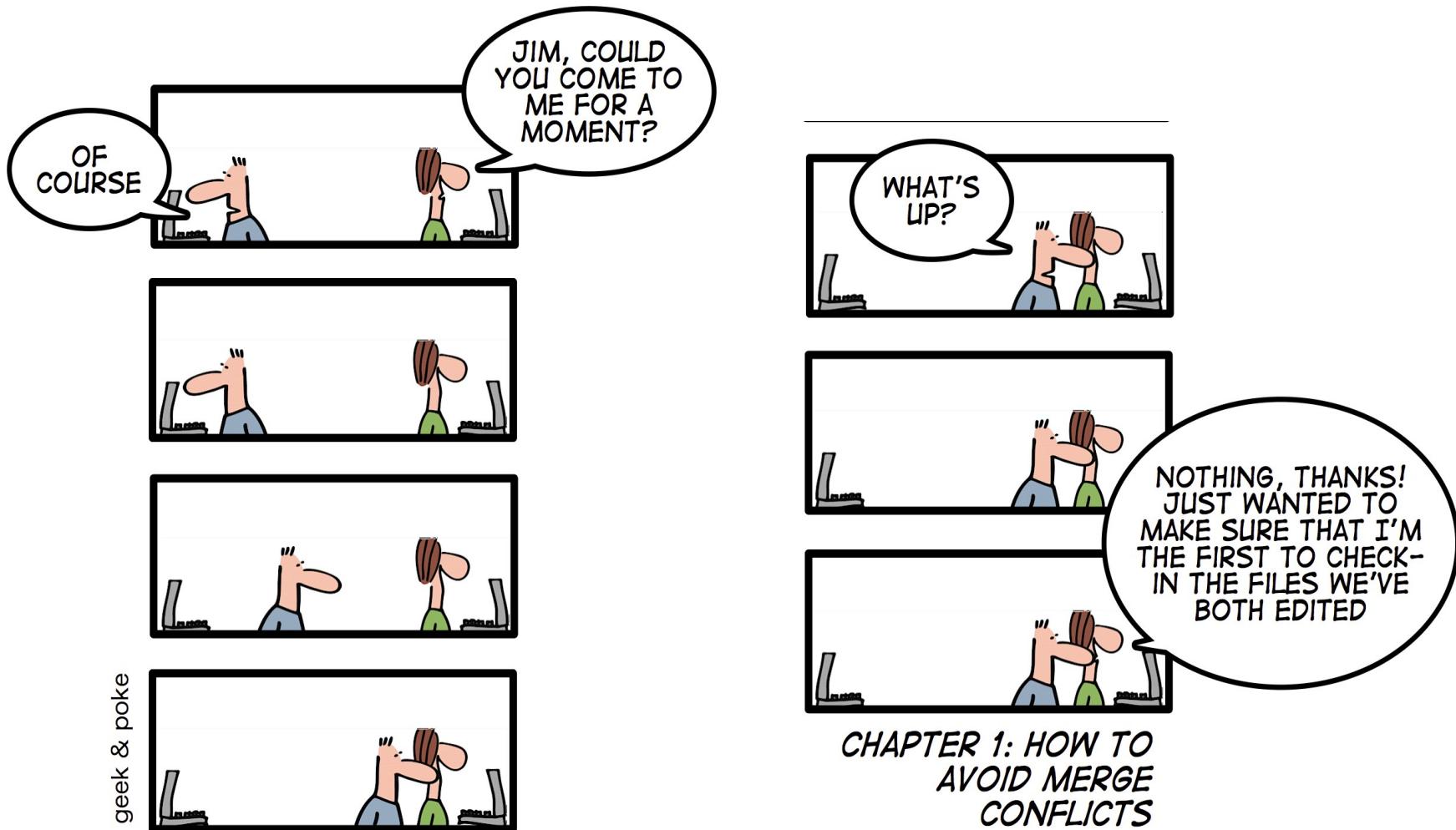
- L'utilisation de la commande **git clone** vous servira également au niveau personnel, si vous devez changer de machine
 - Votre ordinateur n'est plus opérationnel
 - développement à faire urgemment et vous n'avez pas accès à votre ordinateur
 - Le dépôt local et distant sont désynchronisés, vous n'arrivez pas à corriger cet état et vous ne voulez pas perdre vos modifs locales → création d'un autre dépôt local “à jour” dans lequel reporter vos modifications
- Une fois qu'un dépôt central existe, il n'est plus question de faire un **git init** : sa seule utilité est d'initier un dépôt avec des fichiers existants avant d'effectuer un git push vers le dépôt central
- Dès qu'un dépôt central existe, on utilisera **git clone**

Démo 4 : conflits

- Il est possible de créer des conflits à partir du moment où un collaborateur effectue une modification sur les mêmes fichiers que vous et que vous possédez le même ancêtre commun
- Le premier développeur à faire un “push” verra ses modifications prises compte sur le dépôt central
- Le deuxième, en synchronisant son dépôt local verra apparaître des conflits qui devront être résolus manuellement
- Ce n'est qu'après la résolution qu'il pourra faire un “push”.... si personne entre temps n'a mis à jour le dépôt central ^^

Démo 4 : conflits

- Astuce pour éviter les conflits quand vous travaillez à plusieurs



Démo 4 : conflits

- Sur les 2 dépôts locaux, on modifie le fichier main.c en rajoutant un printf() différent à chaque fois
- Pour chaque dépôt local, on fait un **git commit**, suivi par un **git pull** (pour vérifier qu'il n'y a pas de modifications sur le central)
- **DEV #2 → pull / push**

```
cytech@student-laptop:~/tmp/git002$ git pull
Already up to date.
cytech@student-laptop:~/tmp/git002$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done. modifie le fichier main.c en
Writing objects: 100% (3/3), 449 bytes | 449.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0) on fait un commit, suivi par un gi
To https://github.com/Romuald78/testgit002
  c5f2aa5..f98857e main -> main
cytech@student-laptop:~/tmp/git002$
```

Démo 4 : conflits

- Sur les 2 dépôts locaux, on modifie le fichier main.c en rajoutant un printf() différent à chaque fois
- Pour chaque dépôt local, on fait un **git commit**, suivi par un **git pull** (pour vérifier qu'il n'y a pas de modifications sur le central)
- **DEV #2 → log**

The screenshot shows a terminal window with the following content:

```
cytech@student-laptop:~/tmp/git002$ git log -p -1
commit f98857ed7e4c229e6692c88ca8a8e4c0abc6db74 (HEAD -> main, origin/main, origin/HEAD)
Author: Romuald GRIGNON <romuald.grignon@gmail.com>
Date: Mon Apr 18 15:53:56 2022 +0200

Modif dev#2 Writing objects: 100% (3/3), 449 bytes | 449.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/Romuald78/testgit002
diff --git a/main.c b/main.c
index 6861701..9c8270a 100644
--- a/main.c
+++ b/main.c
@@ -4,5 +4,8 @@
 int main(){
     printf("Hello world !\n");
     printf("PI = %f\n", PI);
+
+    printf("Ceci est la modif du DEV #2 avec un saut de ligne avant et après\n");
+
     return 0;
 }
```

The terminal window has a dark background and light-colored text. It includes a top menu bar, a status bar at the bottom, and a right-hand sidebar with font and color selection tools.

Démo 4 : conflits

- Sur les 2 dépôts locaux, on modifie le fichier main.c en rajoutant un printf() différent à chaque fois
- Pour chaque dépôt local, on fait un **git commit**, suivi par un **git pull** (pour vérifier qu'il n'y a pas de modifications sur le central)
- **DEV #1 → pull ! Echec de la fusion automatique**



```
cytech@student-laptop:~/tmp/git001$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 429 bytes | 429.00 KiB/s, done.
From https://github.com/Romuald78/testgit002
  c5f2aa5..f98857e main > origin/main
Auto-merging main.c
CONFLICT (content): Merge conflict in main.c
Automatic merge failed; fix conflicts and then commit the result.
cytech@student-laptop:~/tmp/git001$
```

Démo 4 : conflits

- Il faut ouvrir le fichier main.c qui a été modifié par git et résoudre le conflit manuellement
- Les modifications locales sont entre les “<<<<” et les “=====”
- Les modifications distantes en conflits sont entre les “=====” et les “>>>>”
- Chaque train de chevrons est suivi de la signature du commit



```
cytech@student-laptop:~/tmp/git001$ cat main.c
#include <stdio.h>
#include "constants.h"
5   printf("Hello world !\n");
6   PI = %f\n", PI);
7 int main(){
8   printf("Hello world !\n");
9   printf("PI = %f\n", PI);  
t de ligne avant et un autre printf après\n");
10  <<<<< HEAD  
ème printf du dev #1\n");
11
12  printf("Modif DEV#1 avec un saut de ligne devant et un autre printf après\n");
13  printf("2ème printf du dev #1\n");
14  ===== f98857ed7e4c229e6692c88ca8a8e4c0abc6db74
15  return 0;
16
17  printf("Ceci est la modif du DEV #2 avec un saut de ligne avant et après\n");
18  >>>>> f98857ed7e4c229e6692c88ca8a8e4c0abc6db74
19      return 0;
20 }
```

cytech@student-laptop:~/tmp/git001\$

Démo 4 : conflits

- Nous allons conserver les 3 printf() et supprimer les 2 lignes vides comme résultat de la fusion
- Cette opération est forcément manuelle car l'outil ne peut pas deviner quel est le fonctionnel attendu sur le code

```
cytech@student-laptop:~/tmp/git001$ cat main.c
#include <stdio.h>
#include "constants.h"
5   printf("Hello world !\n");
6   printf("PI = %f\n", PI);
7   printf("Modif DEV#1 avec un saut de ligne avant et un autre printf après\n");
8   printf("Hello world !\n");
9   printf("PI = %f\n", PI); #2 avec un saut de ligne avant et après\n");
10  printf("Modif DEV#1 avec un saut de ligne avant et un autre printf après\n");
11  printf("2ème printf du dev #1\n");
12  printf("Ceci est la modif du DEV #2 avec un saut de ligne avant et après\n");
13  return 0;
}
cytech@student-laptop:~/tmp/git001$ 
```

Démo 4 : conflits

- Nous pouvons effectuer le commit en indiquant que c'est un commit de fusion
- Il faudra bien sur utiliser la séquence classique **git add / git commit**

```
cytech@student-laptop: ~/tmp/git001$ git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
(use "git pull" to merge the remote branch into yours)

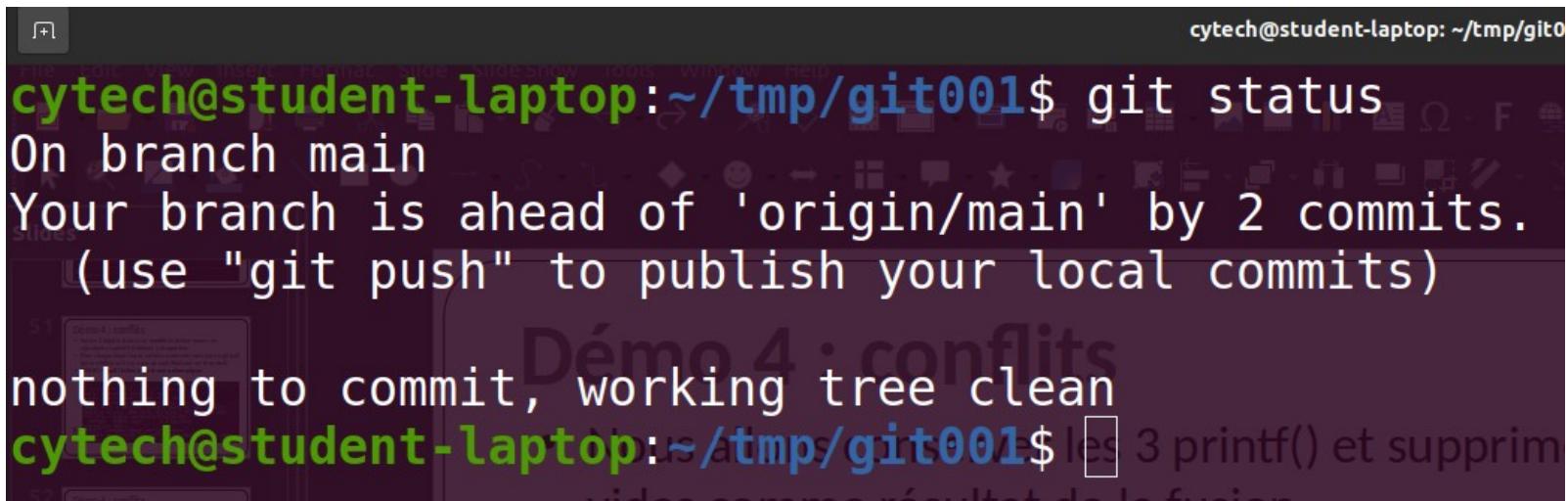
You have unmerged paths. (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified: main.c
cytech@student-laptop: ~/tmp/git001$ cat main.c
#include <stdio.h>
#include "content.h"
printf("Hello world !\n");
int f( int a, b ) { return a + b; }

no changes added to commit (use "git add" and/or "git commit -a")
cytech@student-laptop: ~/tmp/git001$
```

Démo 4 : conflits

- Après le commit de notre résolution de conflit, notre dépôt local est en avance de 2 commits par rapport au dernier état connu du dépôt distant



A screenshot of a terminal window titled "Démo 4 : conflits". The terminal shows the following output:

```
cytech@student-laptop:~/tmp/git001$ git status
On branch main
Your branch is ahead of 'origin/main' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
cytech@student-laptop:~/tmp/git001$ les 3 printf() et supprim
vidéo comme résultat de la fusion.
```

The terminal window has a dark background with light-colored text. The title bar says "Démo 4 : conflits". The command "git status" is run, showing that the local branch "main" is ahead of the remote branch "origin/main" by 2 commits. It also indicates there is nothing to commit and the working tree is clean. The bottom part of the terminal shows some partially visible text related to file operations.

- On peut effectuer un **git pull** pour vérifier qu'un autre développeur n'a pas modifié le dépôt central
- Si il n'y a pas de conflit, on peut faire un **git push**

Démo 4 : conflits

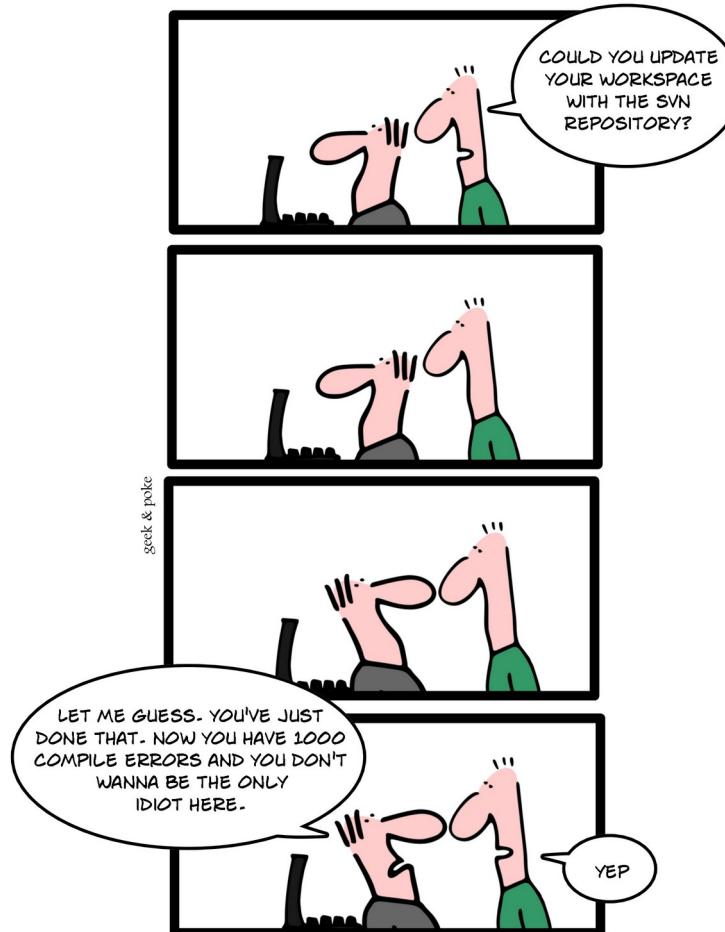
- Après la commande git push réussie, l'état du dépôt central est bien le résultat de la fusion des 2 développeurs : le **DEV #1** pourra récupérer cette fusion avec git pull

```
cytech@student-laptop:~/tmp/git001$ git push
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 715 bytes | 715.00 KiB/s, done.
Total 6 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/Romuald78/testgit002.git
  f98857e..3ee641e main -> main
cytech@student-laptop:~/tmp/git001$
```

Démo 4 : conflits

- Se mettre à jour régulièrement par rapport au dépôt central permet d'éviter des fusions trop importantes/pénibles ...

REAL CODERS HELP EACH OTHER



... ou pas

Démo 4 : conflits

- Si la fusion est complexe elle peut nécessiter plusieurs développeurs pour en venir à bout : chacun ayant la connaissance d'une partie de la modification
- Le fait de séparer le code en modules permet d'éliminer des fusions et/ou réduire leur complexité
- Malgré cette fusion, nous ne travaillons pas ici dans plusieurs branches mais bien une seule : la gestion de plusieurs branches de travail permet également d'éviter (temporairement) des fusions, et surtout de ne pas perturber l'état de l'application

The screenshot shows a Git commit history interface. At the top, there's a menu bar with File, Edit, View, Help. Below the menu, there's a toolbar with a double arrow icon. The main area displays a list of commits. On the left, there's a commit graph showing a merge between 'main' and 'remotes/origin/main'. The 'main' branch has two blue nodes labeled 'Modif dev#2' and 'Modif dev#1'. The 'remotes/origin/main' branch has three purple nodes: 'Revert "ajout de la modification à supprimer"', 'ajout de la modification à supprimer', and 'add constants to the program'. A yellow node at the bottom is labeled 'Premier commit de mon programme C'. To the right of the commits is a detailed log table.

Author	Date
Romuald GRIGNON	2022-04-18 16:16:25
Romuald GRIGNON	2022-04-18 15:53:56
Romuald GRIGNON	2022-04-18 15:54:54
Romuald GRIGNON	2022-04-18 12:47:31
Romuald GRIGNON	2022-04-18 12:35:57
Romuald GRIGNON	2022-04-17 23:36:51
Romuald GRIGNON	2022-04-17 23:11:37

Démo 5 : les branches



Démo 5 : branches

- La commande **git branch <nom>** permet de créer une nouvelle branche (ici spécifique aux développements du *DEV #2*)
- La commande **git checkout <nom>** permet de basculer d'une branche à l'autre



```
cytech@student-laptop:~/tmp/git002$ git branch dev2
cytech@student-laptop:~/tmp/git002$ git branch
  dev2
* main
cytech@student-laptop:~/tmp/git002$ git checkout dev2
Switched to branch 'dev2'
cytech@student-laptop:~/tmp/git002$ git branch
* dev2
  main
cytech@student-laptop:~/tmp/git002$
```

The screenshot shows a terminal window with the following content:

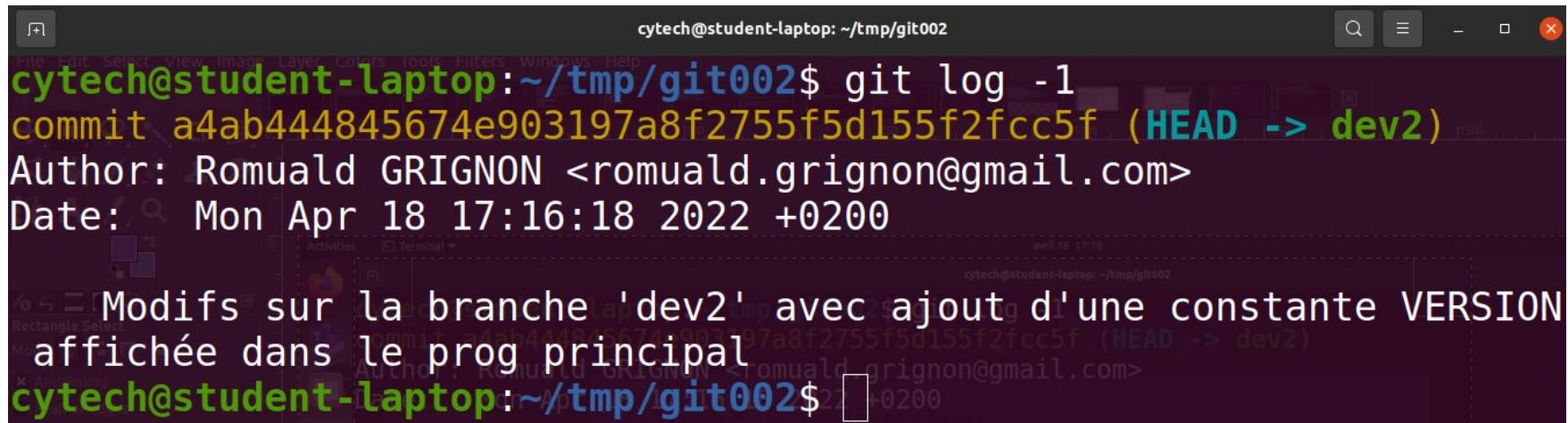
- Terminal title: cytech@student-laptop: ~/tmp/git002
- First command: `git branch dev2`
- Second command: `git branch` (shows branches `dev2` and `main`)
- Third command: `git checkout dev2` (switches to the `dev2` branch)
- Fourth command: `git branch` (shows branches `* dev2` and `main`)

The background of the terminal window shows a blurred image of a supermarket aisle.

- Nous allons modifier les fichiers `main.c` et `constants.h` dans la branche `dev2` et effectuer ce commit

Démo 5 : branches

- Après les modifications et le commit dans la branche, nous avons un état stable en local



A screenshot of a terminal window titled "cytech@student-laptop: ~/tmp/git002". The window shows the command "git log -1" being run. The output is:

```
commit a4ab444845674e903197a8f2755f5d155f2fcc5f (HEAD -> dev2)
Author: Romuald GRIGNON <romuald.grignon@gmail.com>
Date: Mon Apr 18 17:16:18 2022 +0200

    Modifs sur la branche 'dev2' avec ajout d'une constante VERSION
    affichée dans le prog principal
```

The terminal window has a dark background and light-colored text. It is part of a desktop environment with icons for Activities, Terminal, and other applications visible.

- Mais il faut dire au dépôt distant de suivre cette nouvelle branche également
- Comme avec le dépôt vide au début du cours, en utilisant l'option -u de git push (lors du premier push uniquement)

Démo 5 : branches

- Le push de la nouvelle branche sur le dépôt distant

```
cytech@student-laptop:~/tmp/git002$ git push -u origin dev2
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 568 bytes | 568.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
remote:         avons un état stable en local
remote: Create a pull request for 'dev2' on GitHub by visiting:
remote:     https://github.com/Romuald78/testgit002/pull/new/dev2
remote:
To https://github.com/Romuald78/testgit002
 * [new branch]      dev2 -> dev2
Branch 'dev2' set up to track remote branch 'dev2' from 'origin'.
cytech@student-laptop:~/tmp/git002$
```

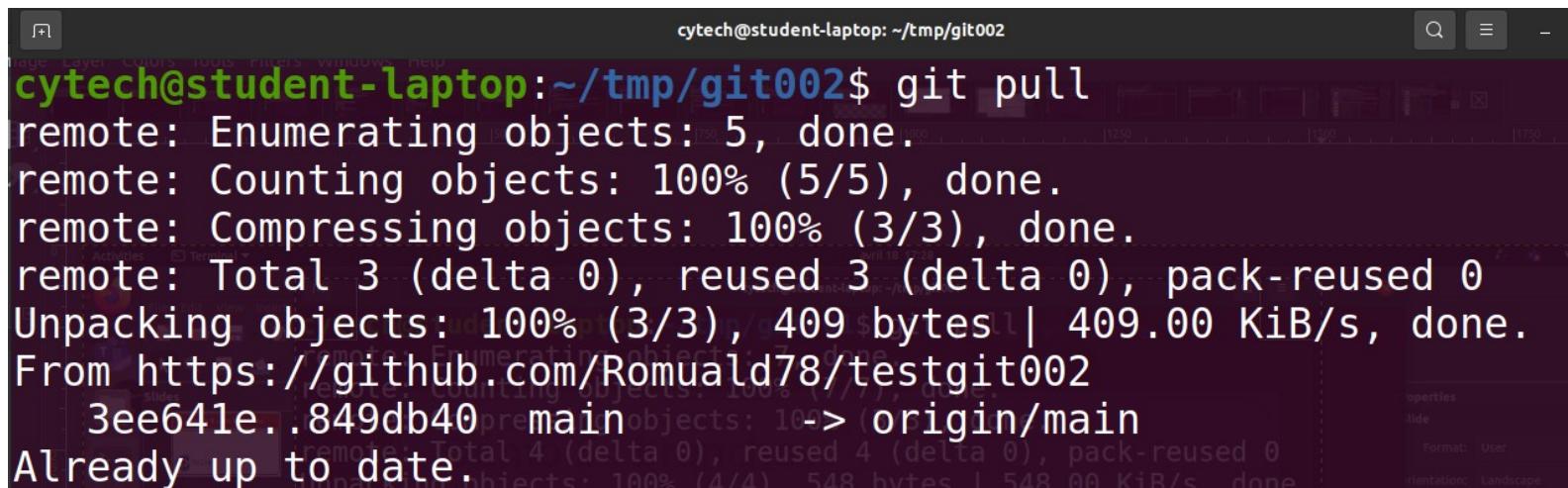
Démo 5 : branches

- La récupération par le développeur **DEV #1** de cette branche ‘dev2’ ne crée aucun conflit puisqu’il est toujours sur la branche ‘main’

```
cytech@student-laptop:~/tmp/git001$ git pull
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 4 (delta 0), pack-reused 0
Unpacking objects: 100% (4/4), 548 bytes | 548.00 KiB/s, done.
From https://github.com/Romuald78/testgit002
 * [new branch]      dev2    -> origin/dev2
Already up to date.
cytech@student-laptop:~/tmp/git001$
```

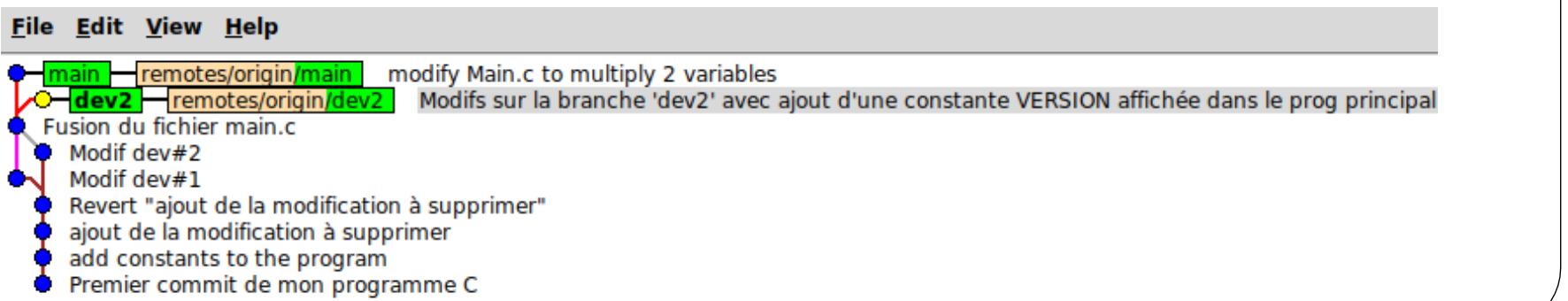
Démo 5 : branches

- Si le développeur **DEV #1** effectue une modification du fichier main.c sur la branche principale et commit, le **DEV #2** pourra récupérer ces modifications sans conflit avec **git pull**



```
cytech@student-laptop: ~/tmp/git002$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 409 bytes | 409.00 KiB/s, done.
From https://github.com/Romuald78/testgit002
  3ee641e..849db40 main    -> origin/main
Already up-to-date.
```

- On voit les 2 branches qui divergent avec l'ancêtre commun

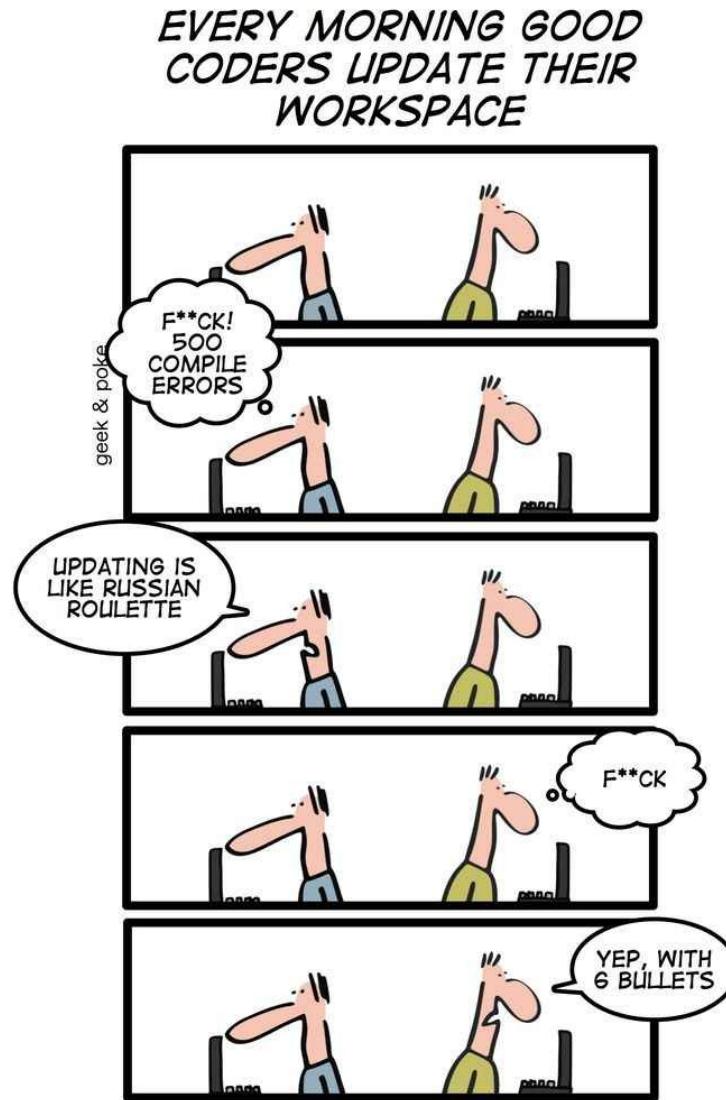


Démo 5 : branches

- Pour fusionner la branche de développement dans la branche principale, il est préférable de plutôt faire l'inverse dans un premier temps
- Donc, rappatrier la branche principale dans la branche de développement : seule cette dernière sera impactée, à charge au(x) développeur(s) de stabiliser cette branche (résoudre les conflits, passer les tests, ...)
- Une fois fait, la fusion vers la branche principale sera automatique. Les évolutions de la branche de développement n'ont que peu d'impact sur le travail des autres équipiers, surtout si cette branche de développement vit longtemps
- Il est donc primordial pour un développeur de se mettre à jour régulièrement par rapport à la branche principale

Démo 5 : branches

- L'idéal étant de mettre à jour son poste de travail tous les jours



Démo 5 : branches

- On se place dans la branche de développement ‘dev2’
- On utilise la commande **git merge <nom>** pour rappatrier les modifications d'une autre branche (ici ‘main’)



A screenshot of a terminal window titled "cytech@student-laptop: ~/tmp/git002". The command "git merge main" is run, resulting in a merge conflict in the file "main.c". The terminal shows:

```
cytech@student-laptop:~/tmp/git002$ git merge main
Auto-merging main.c
CONFLICT (content): Merge conflict in main.c
Automatic merge failed; fix conflicts and then commit the result.
cytech@student-laptop:~/tmp/git002$
```

- Bien évidemment il y a des conflits dans le fichier main.c (pas dans constants.h puisqu'il n'y a qu'un rajout)
- Ces conflits n'impactent personne parmi les autres membres de l'équipe : c'est le but des branches (même pour un projet perso, le fait de tester une fonctionnalité en branche, permet de ne pas casser le reste de l'application)

Démo 5 : branches

- Le fichier main.c a donc un conflit : nous allons choisir de garder tous les printf() ainsi que les déclarations des 2 variables

The screenshot shows a terminal window with the following content:

```
cytech@student-laptop:~/tmp/git002$ cat main.c
#include <stdio.h>
#include "constants.h"

int main(){
<<<<< HEAD
    printf("Hello world !\n");
    printf("PI = %f\n", PI);
    printf("MODIFICATION de la branche 'dev2'\n");
    printf("[version = %s]\n", VERSION);

=====
    int a = 5;
    int b = 7;
    printf("Produit de %d et %d = %d \n", a, b, a*b);
>>>>> main
        return 0;
}
```

The terminal shows a merge conflict in the `main.c` file. The code contains two sets of `printf` statements: one from the `HEAD` branch and one from the `dev2` branch. A conflict marker (`=====`) separates them. The `git merge` command was run, resulting in a conflict in `main.c`. The terminal prompt is `cytech@student-laptop:~/tmp/git002$`.

Démo 5 : branches

- On se place dans la branche de développement
- On utilise la commande `git merge <nom>` pour fusionner la branche `dev2` dans la branche `main`
- Puis évidemment il y a des conflits dans le fichier `main.c` (dans constants.h puisqu'il n'y a qu'un rajout)
-

Démo 5 : branches

- Le fichier main.c a donc un conflit : nous allons choisir de garder tous les printf() ainsi que les déclarations des 2 variables

The terminal window shows the following content:

```
cytech@student-laptop:~/tmp/git002$ cat main.c
#include <stdio.h>
#include "constants.h"

int main(){
    int a = 5;
    int b = 7;
    printf("Hello world !\n");
    printf("PI = %f\n", PI);
    printf("MODIFICATION de la branche 'dev2' ");
    printf("[version = %s]\n", VERSION);
    printf("Produit de %d et %d = %d \n", a, b, a*b);
    return 0;
}

cytech@student-laptop:~/tmp/git002$
```

Commit history (partial):

- 63: Démo 1 : branches
- 64: Démo 2 : branches
- 65: Démo 3 : branches

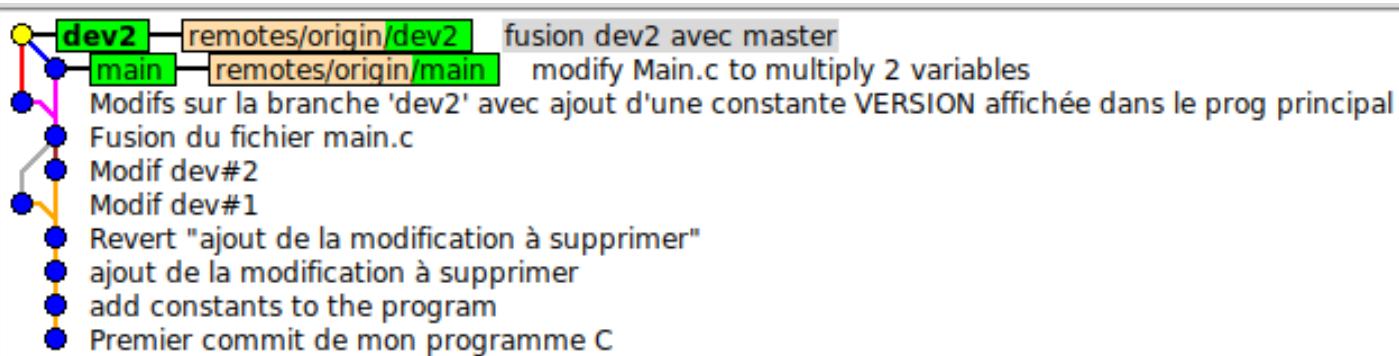
On the right side of the slide, there is a large watermark-like text "Démo 5 : branches" and a list of bullet points:

- Le fichier main.c a donc un conflit : nous allons garder tous les printf() ainsi que les déclarations des 2 variables

Démo 5 : branches

- On ajoute le fichier main.c au prochain commit, on effectue le commit et on synchronise avec le dépôt central

```
cytech@student-laptop:~/tmp/git002$ git add main.c
cytech@student-laptop:~/tmp/git002$ git commit -m "fusion dev2 avec master"
[dev2 8b7f0f5] fusion dev2 avec master
cytech@student-laptop:~/tmp/git002$ git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 435 bytes | 435.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Romuald78/testgit002
  a4ab444..8b7f0f5  dev2 -> dev2
cytech@student-laptop:~/tmp/git002$
```



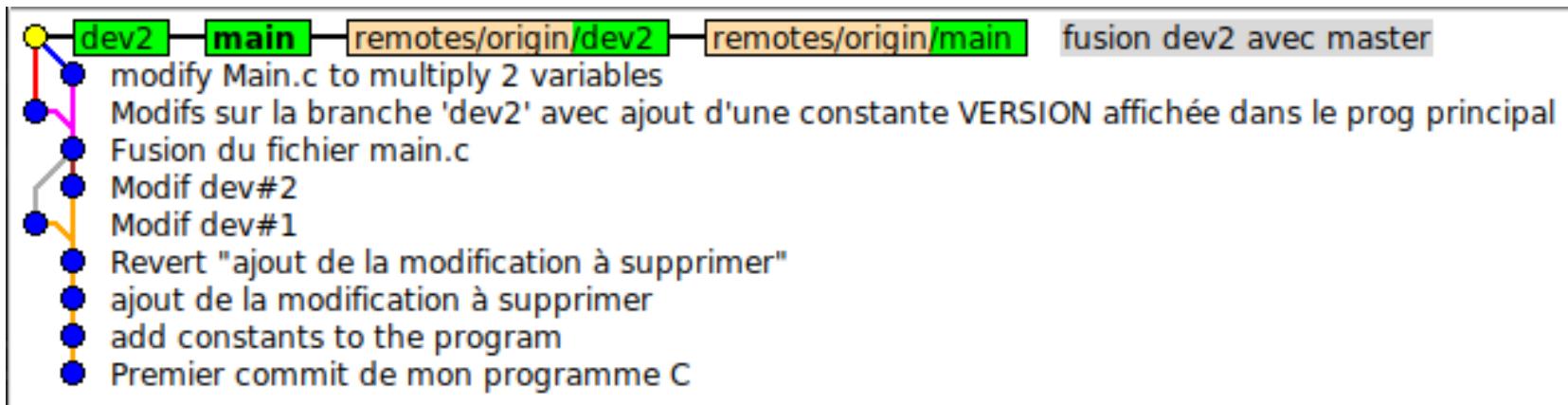
Démo 5 : branches

- Maintenant nous pouvons effectuer l'opération ultime : soumettre les modifications de notre branche de développement dans la branche principale

```
cytech@student-laptop: ~/tmp/git002$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
cytech@student-laptop: ~/tmp/git002$ git merge dev2
Updating 849db40..8b7f0f5
Fast-forward
  constants.h | 1 +
  main.c      | 8 +++++- -
  2 files changed, 6 insertions(+), 3 deletions(-)
cytech@student-laptop: ~/tmp/git002$ git push
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/Romuald78/testgit002
  849db40..8b7f0f5  main -> main
cytech@student-laptop: ~/tmp/git002$
```

Démo 5 : branches

- Le commit de fusion de la branche ‘dev2’ devient la nouvelle révision de tête pour la branche ‘main’
- Toute l’équipe peut maintenant profiter des fonctionnalités du DEV #2 (et tous les bugs qu’il aura créé aussi)



Interfaces graphiques

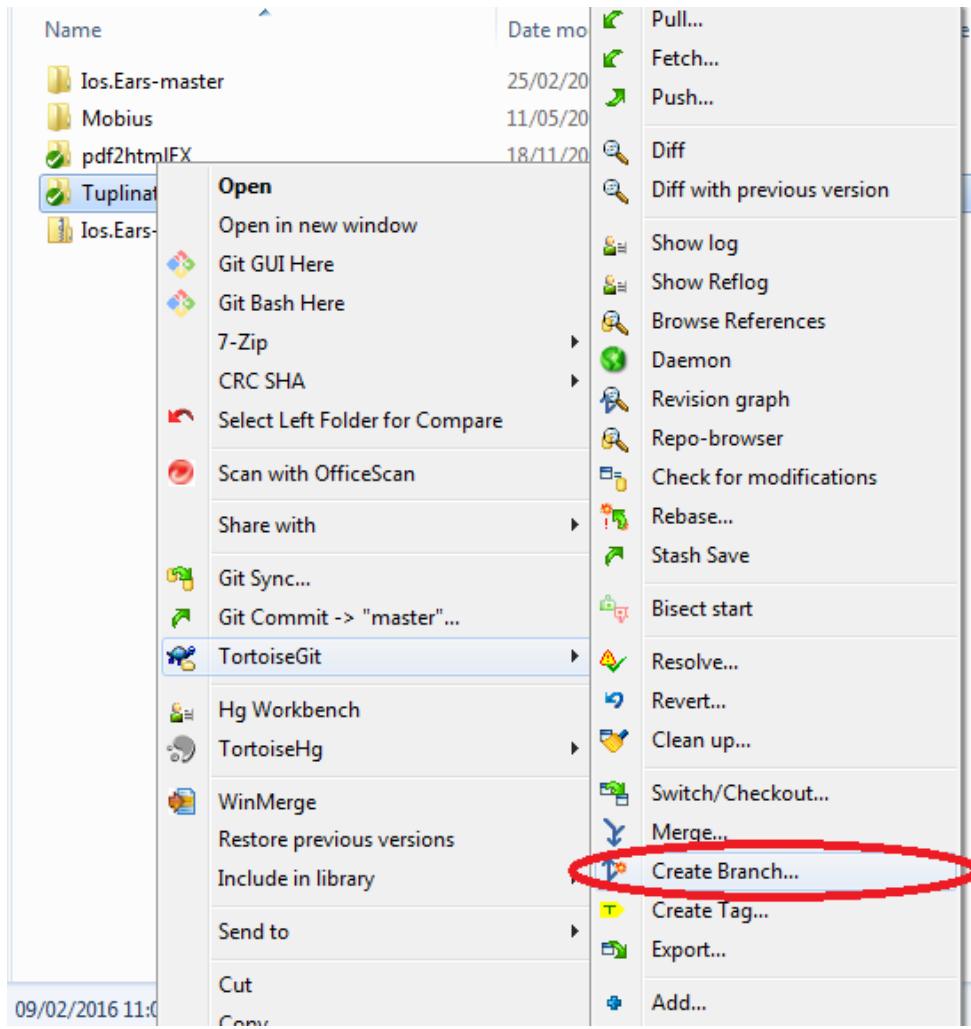


Interfaces graphiques

- Pour plus d'ergonomie, certains logiciels fournissent une interface graphique à un projet Git
- Certains sont directement fournis avec Git (ou presque, comme gitk donc vous avez vu quelques captures d'écran dans ce document)
- Développés par des entités tierces
- Intégrés dans des environnements de développement (comme ceux de chez JetBrains : PyCharm, IntelliJ, ...)
- Permet d'avoir une meilleure visibilité des différents commits, de l'enchevêtrement des branches, etc...

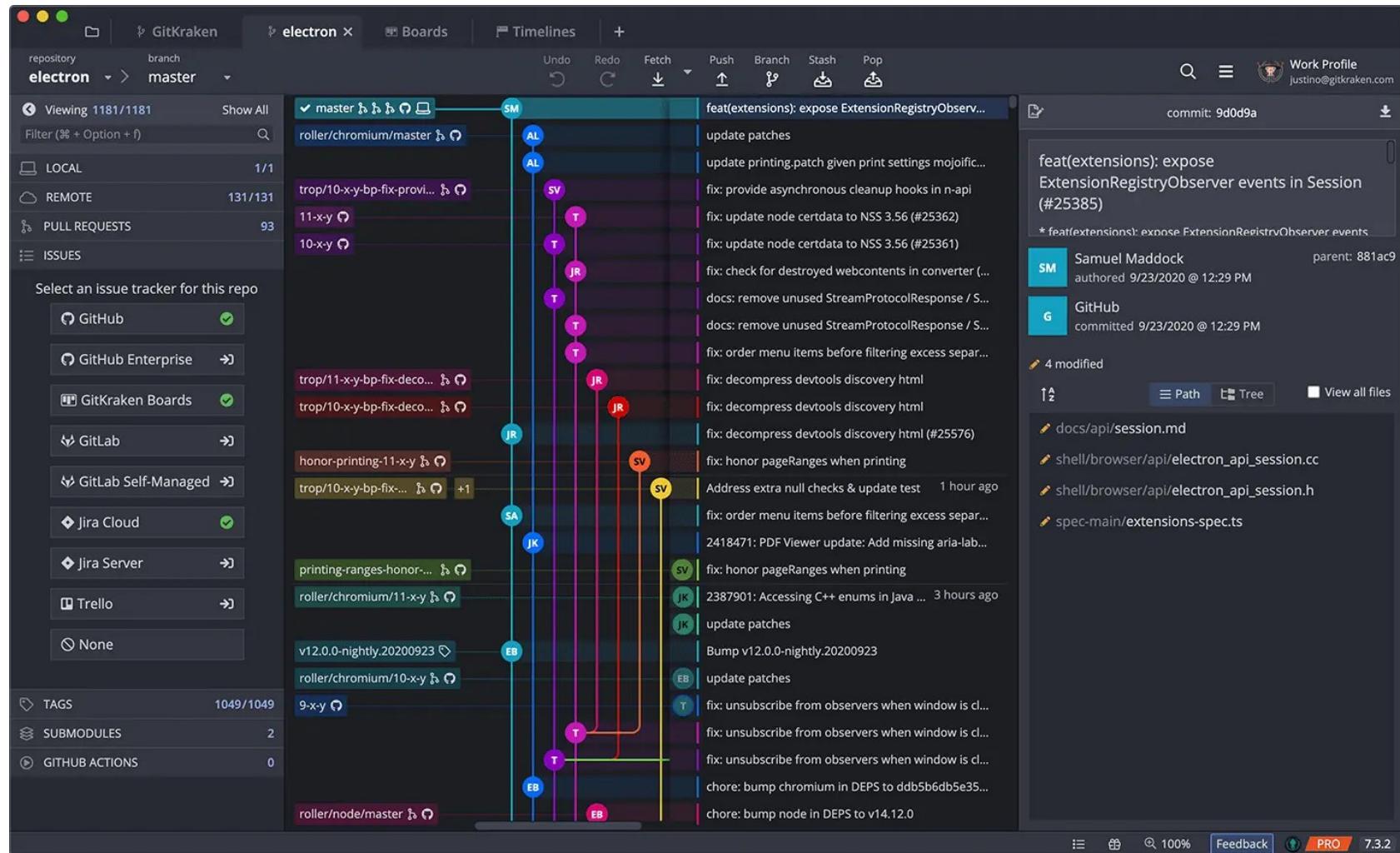
Interfaces graphiques

- TortoiseGit sous Windows permet d'avoir toutes les commandes git dans le menu contextuel clic-droit



Interfaces graphiques

- GitKraken, multi-plateformes, possède une interface graphique agréable et ergonomique



Conclusion



Conclusion

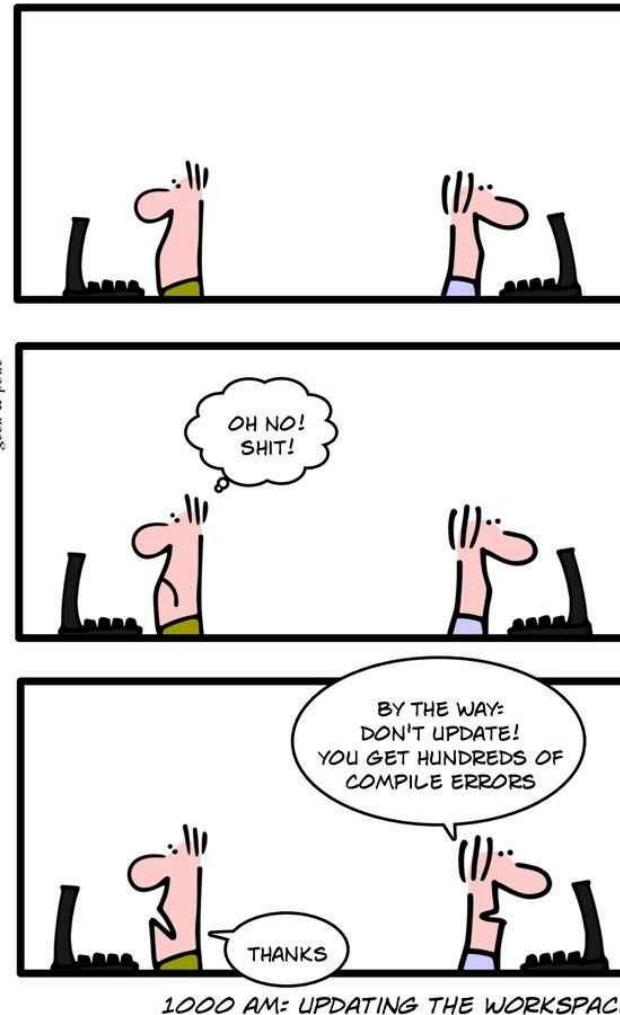
- Les outils de gestion de versions sont des outils puissants permettant de garantir la traçabilité de votre code, et de travailler plus efficacement en équipe
- C'est devenu un outil indispensable à tout développeur professionnel : sa connaissance est tout aussi importante que le code lui-même
- Même si nous n'avons vu que Git, il existe une multitude d'autres outils plus ou moins équivalents
 - Propriétaires ou libres
 - Centralisés ou non
 - Avec des clients graphiques ou en ligne de commande
 - Des plateformes web compatibles pour l'hébergement

Conclusion

- Même si ils restent des outils incontournables, leur utilisation reste sensible
- Il n'est pas possible de perdre des informations mais les retrouver peut s'avérer complexe si on ne fait pas attention aux commandes que l'on exécute
- La connaissance des différentes commandes est un plus pour l'utilisation des interfaces graphiques
- Rien que pour Git, le nombre d'options pour les différentes commandes est énorme. La maîtrise d'un tel outil peut prendre du temps, surtout si l'on s'attarde au cœur de l'exécutable et la manière dont il stocke chaque révision

Conclusion

- Les outils de gestion de configuration ne remplaceront jamais le dialogue entre équipiers ...



Conclusion

- ... et seront inutiles face à des méthodes de développement peu scrupuleuses :

DEVELOPMENT CYCLE

FRIDAY EVENING EDITION

