


La gestion de version dans les Projets Informatiques



Gestion de versions

- Introduction à cette formation
 - Votre formateur ... Et Vous 
 - Le matériel
 - Le support de cours
 - Les stations Windows
 - L'organisation – horaires
 - Formation de 1.5 jour 9h 12h30 et 13h30 17h
 - La forme :
 - Un mélange de concepts avec application directe par un exemple simple
 - Des exercices

Gestion de versions

- Des liens utiles

- <https://www.g2.com/categories/version-control-software> pour la comparaison d'outils de versionning
- <https://try.github.io> pour découvrir GIT
- <https://book.git-scm.com/docs> doc officielle
- <https://book.git-scm.com/book/en/v2> livre
- <https://openclassrooms.com/courses/gerer-son-code-avec-git-et-github>

- Problématique de la gestion de version
- La gestion de configuration ?
- Les types d'outil de gestion de version
- Présentation et utilisation de Git
- Présentation et utilisation de GitHub
- Quelques WorkFlow avec Git
- Présentation et utilisation de TurtoiseGit
- Utilisation de Visual Studio Code



GitHub



Projet en développement

Qui a modifié le
fichier X ?

Fichiers
sources

C#, Java, SQL,
CSS, HTML ...

Fichiers
configuration

xml, vcproj, txt ...

Comment
travailler en
équipe ?

Qui a ajouté
cette ligne ?



s
a
?

- Un projet informatique consiste en l'écriture de fichiers sources :
 - qui évoluent dans le temps
 - Correction d'anomalies
 - Nouvelles fonctionnalités demandées
 - Fonctionnalité(s) supprimée(s)
 - Fonctionnalité(s) reprise(s) : refonte du code
 - Résultat du travail de développeurs différents
 - Plusieurs versions bien identifiées du même fichier sont nécessaires. Il faut pouvoir utiliser la bonne version pour re générer une version précédente de l'application

- Les outils actuels de gestion de versions
 - Gardent l'évolution des fichiers source
 - Identifient qui a effectué les modifications et pour quelles raisons



Versions précédentes

Fichier source dernière version

- Dans le contexte de travail en groupe, permettent le travail simultané sur les mêmes fichiers, par le mécanisme de branche et de fusion (merge)



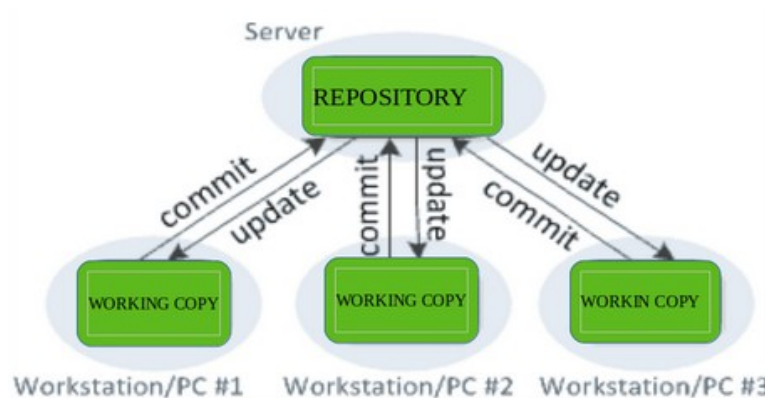
Imie-paris

- La gestion de Versions permet de connaître, pour une version donnée de l'application, le code source exact à utiliser pour générer cette application
- La gestion de Configuration :
 - Contient les éléments de la gestion de versions
 - Décrit le contexte dans lequel l'application doit fonctionner :
 - Le matériel cible (station, carte microcontrôleur ...)
 - L'OS ou le noyau de la machine cible (Linux, MacOS, VRTX ...)
 - Les versions compatibles des logiciels tierces (ORACLE, Qt ...)
 - Produit une fiche version ou document de version contenant la liste des anomalies et évolutions qu'apporte cette version
 - Utilise un outil de suivi des problèmes

La gestion de Configuration n'est pas détaillée dans ce cours

Gestion de versions

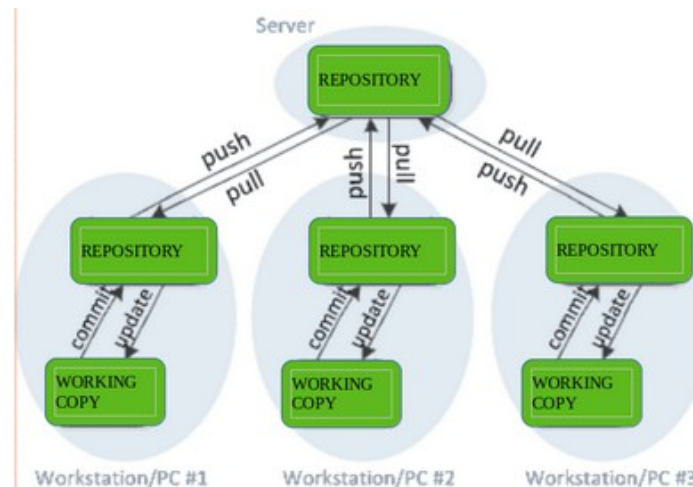
- Il existe deux catégories d'outils de gestion de versions :
 - Logiciels centralisés : un serveur contient et centralise les données propres à la gestion : toutes les versions des fichiers, contextes et modifications des utilisateurs.
Les développeurs se connectent pour éditer et modifier les fichiers.



- Avantage : les développeurs voient rapidement les modifications des collègues
- Inconvénient : la perte du serveur empêche le travail

Gestion de versions



- Logiciels distribués : chaque poste développeur possède l'historique et l'ensemble des fichiers. La communication de modifications se fait entre les développeurs, en liaison point à point.



Comment choisir ?

<https://www.geeksforgeeks.org/centralized-vs-distributed-version-control-which-one-should-we-choose/>

Gestion de versions

- Les freewares de gestion de version les plus connus :
 - Subversion (dev Apache), alias SVN remplace sa précédente version appelée CVS. La version Windows avec une interface graphique s'appelle Tortoise SVN
Type Centralisé.
Très utilisé. 
 - Mercurial. Type distribué
 - Bazaar. Type distribué.
 - Git. Type distribué.
Est le support du site de partage GitHub.
TortoiseGit propose une interface sous Windows 

Nous allons découvrir Git





Télécharger et installer Git

<https://git-scm.com/downloads> (2.40.0 sous Windows en 2023)

- Ouvrir le fenêtre console Git Bash
Git fonctionne dans un environnement UNIX/Linux
Bash reproduit sous Windows un environnement UNIX shell
- Configurer Git : fournir un pseudo et une adresse mail
\$ `git config --global user.name "Jamal Hartnett"`
\$ `git config --global user.email "jamal@fabrikam.com"`
- \$ `git config --list` pour voir le résultat

- Créer le 1^{er} espace (repository, dépôt en français) sur le disque local.
Dans la fenêtre Bash saisir : ('\$' signifie le début de ligne (prompt), ne pas le saisir)

`$pwd` montre le répertoire courant
`$mkdir monRepo` crée un répertoire *monRepo*
`$cd monRepo` se déplacer dans ce répertoire
`$pwd`
`$git init` considère monRepo en tant que repository
 (.git créé)
`$git status` état du repository

- Créer le fichier code.txt dans monRepo et écrire deux lignes.
Sauvegarder le fichier
`$git status` code.txt vu mais non géré
- Ajouter ce fichier code.txt dans la gestion de git
`$git add code.txt` Ajoute code.txt à la gestion Git
`$git status` code.txt vu et géré
- Les caractères génériques sont permis
`$git add .` Tout le répertoire
`$git add *.txt` Tous les txt
- Enregistrer cette modification (ajout d'un fichier) : faire un "commit"
L'option `-m` est utilisée pour commenter le commit
`$git commit -m "mon commentaire de commit"`

```
$ git commit -m "ajout de code.txt"
[master (root-commit) 94dd232] ajout de code.txt
1 file changed, 1 insertion(+)
create mode 100644 Code.txt
```

- Regarder l'historique des commit : git log

\$git log

```
$ git log
commit 94dd23205de19bda2d1339efcfe3414e73ecf435 (HEAD -> master)
Author: albert <d.roches@pragma-tec.fr>
Date:   Wed Jun 21 10:10:16 2017 +0200

    ajout de code.txt
```

montre pour chaque commit :

- Le SHA : identifiant unique
- Auteur du commit et date
- Description du commit (-m du git commit -m)
- HEAD désigne l'espace courant de Git
- master désigne la Branche principale de Git (vu ci-après)

- Mettre à jour le fichier code.txt et enregistrer la modif :
 - Editer code.txt, modifier une ligne existante, supprimer l'autre et ajouter une nouvelle ligne. Sauvegarder le fichier.
 - Enregistrer la modif auprès de Git :
`$git commit -a -m "Fichier code.txt modifié"` (ou `-am`)
Le `-a` évite un nouveau `git add` sur code.txt
 - `$git log`
- Retour en arrière
Pour récupérer le code d'un commit : `git checkout <SHACommit>`
 - `$git log` recherche du SHA
 - `$git checkout <SHA>`
 - Editer le fichier code.txt et constater son contenu revenu en arrière.

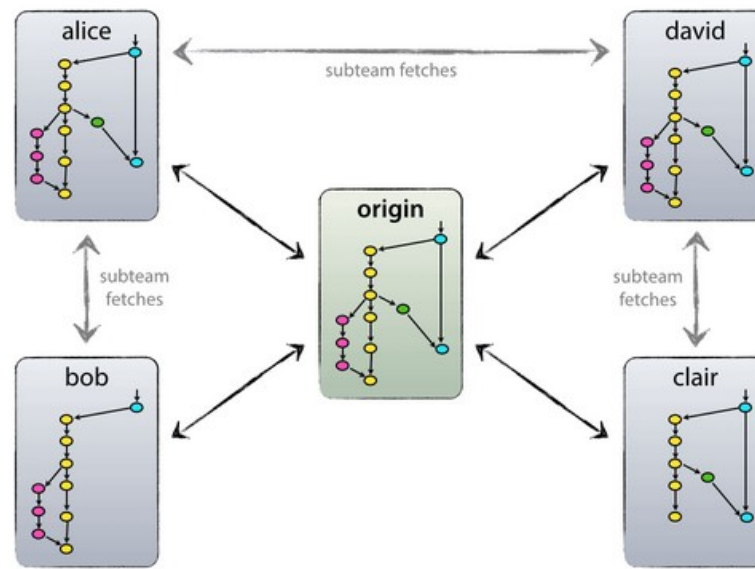
- Retour au commit le plus récent
 - `$git checkout master`
 - Constater que `code.txt` est revenu à son dernier contenu.
- Annuler un commit (on ne peut pas le supprimer)
 - `$git revert <SHACommit>`
- Modifier le message du dernier commit
 - `$git commit --amend -m "nouveau message"`
- Annuler tous les changements avant un commit
 - `$git reset --hard`

- Dans le cadre d'un projet il est essentiel d'avoir un backup du code sur une autre machine.

A partir de commits faits en local sur votre machine, on peut envoyer le résultat sur une machine remote :

- Sur le réseau interne de l'entreprise
- Sur le Web en utilisant des serveurs de type GitHub, GitLab ou BitBucket. Comparaison sur

<https://www.hebergeurcloud.com/github-vs-bitbucket-vs-gitlab-une-bataille-epique-pour-lesprit-de-developpeur/>



- GitHub est un site qui héberge des dépôts ou repositories gérés par Git
- Gratuit pour les repositories public, payant pour les privés
- Propose une ihm pour s'affranchir de la syntaxe des commandes Git
- Il est nécessaire de connaître les principes et utilisation de Git

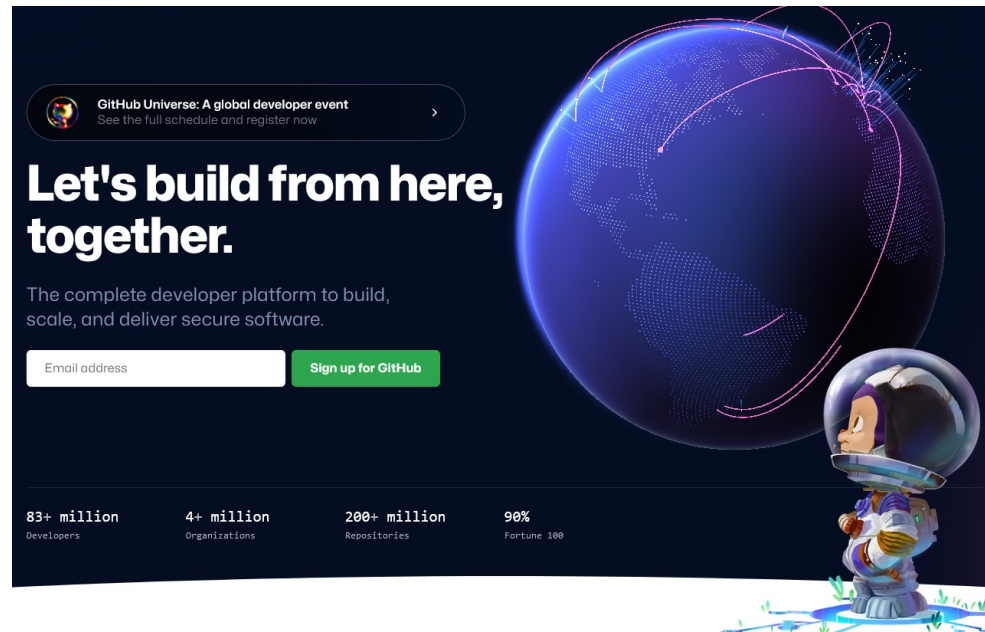




GitHub

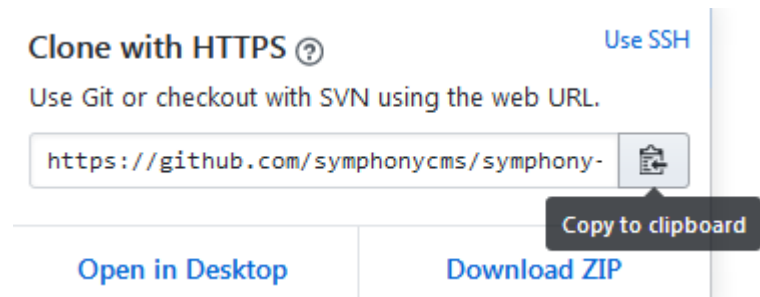
Création d'un compte GITHUB


<https://git-scm.com/book/fr/v2/GitHub-Configuration-et-param%C3%A9trage-d%E2%80%99un-compte>

Inscription sur <https://github.com/>



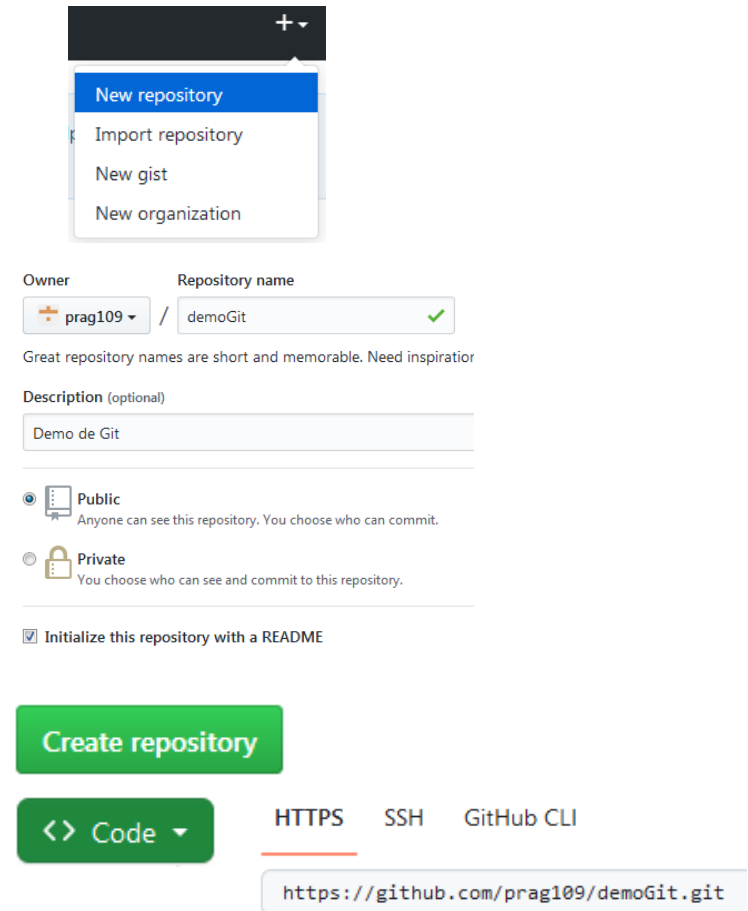
- Récupérer du code d'un autre repository
On souhaite récupérer le code du projet Symfony sur GitHub et le placer sur notre station, géré par Git.
Cette opération s'appelle un Clonage
 - Aller sur le site GitHub <https://github.com/>
 - Clic sur  de la barre de titre et saisir Symfony
 - Clic sur `Symfony/symfony`
 - Explorer la page
 - Clic sur 



- À ce stade l'appui précédent sur  a copié l'adresse (URL) du repository GitHub
- Revenir dans la fenêtre console Bash
 - `$cd ..` Remonter d'un cran dans le chemin
 - `$pwd` montre où l'on est
 - `$mkdir gitdemo` crée le répertoire gitdemo
 - `$cd gitdemo`
 - `$git clone https://github.com/symfony/symfony.git`
Utiliser le Coller pour l'URL
- Le code source de cette version est maintenant disponible localement pour analyse ou modifications.

Créer un repository sur GitHub et le lier à un espace local sur votre station :

- Créer un repository sur GitHub :
 - Depuis votre compte GitHub
 - Nom : demoGit
 - Public
 - README
 - Créer le repository
 - Copier l'URL de ce repository. Clic sur




The screenshot shows the GitHub 'New repository' form. At the top, a dropdown menu is open with options: 'New repository' (highlighted), 'Import repository', 'New gist', and 'New organization'. Below this, the 'Owner' is set to 'prag109' and the 'Repository name' is 'demoGit' with a green checkmark. A note states: 'Great repository names are short and memorable. Need inspiration'. The 'Description (optional)' field contains 'Demo de Git'. Under 'Visibility', 'Public' is selected with the subtext 'Anyone can see this repository. You choose who can commit.' The 'Private' option is also visible with the subtext 'You choose who can see and commit to this repository.' A checkbox 'Initialize this repository with a README' is checked. A large green 'Create repository' button is prominent. Below it, a 'Code' button with a green icon is shown. To the right, three tabs are visible: 'HTTPS' (selected), 'SSH', and 'GitHub CLI'. The 'HTTPS' tab shows the URL 'https://github.com/prag109/demoGit.git' in a text box.

- Lier le repository GitHub à un espace local
Dans la fenêtre Bash
 - `$cd ..`
 - `$pwd`
 - `$git clone https://github.com/prag109/demoGit.git`

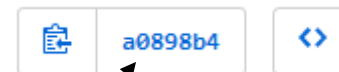
Le répertoire demoGit est alors copié localement et est lié à celui de GitHub. Et c'est aussi un repository.

Le repository distant est nommé "Origin" par Git

Le repository local correspond au nom de la branche (master pour l'instant)

- Faire des modifications locales et les publier sur GitHub :
 - Éditer et modifier README.md (l'éditeur vim UNIX ...)
 - `$git add README.md` ajoute le fichier à Git
 - `$git commit -m "README.md modifié"`
 - `$git push origin main` “pousse” le code de la branche courante locale main (et pas master) vers la destination “origin” i.e GitHub
 - GitHub va demander une authentification par 
 - Constater sur GitHub la présence de 2 commits et regarder la différence

- Seconde modification en local
 - Ajouter le fichier code.txt avec quelques lignes. Sauvegarder.
 - `$git add code.txt`
 - `$git commit -m "ajout de code.txt"`
 - `$git push origin main`
 - Constater sur GitHub la présence de 3 commits
- Dans l'onglet GitHub Commits, l'appui sur le bouton SHA fournit une page montrant les différences



Cliquer ici

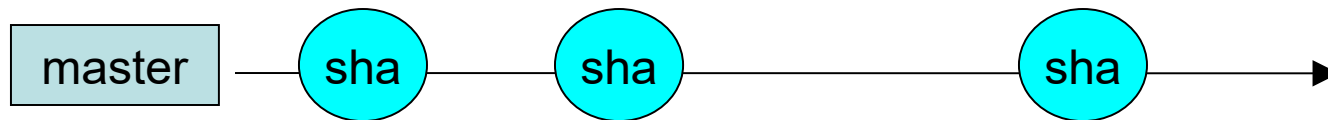
- Récupérer en local des modifications disponibles sur GitHub
 - Ces modification ont été faites directement sous GitHub
 - Ou plus vraisemblable un collègue a mis à disposition ses modifications depuis son poste local (git push origin main)

Simulons-le :

- Depuis GitHub modifier code.txt en modifiant la ligne 2 ajouter une ligne
- Depuis GitHub faire un commit avec commentaire
- Sur le poste client, console Bash :
- `$git pull origin main`
- Vérifier le changement

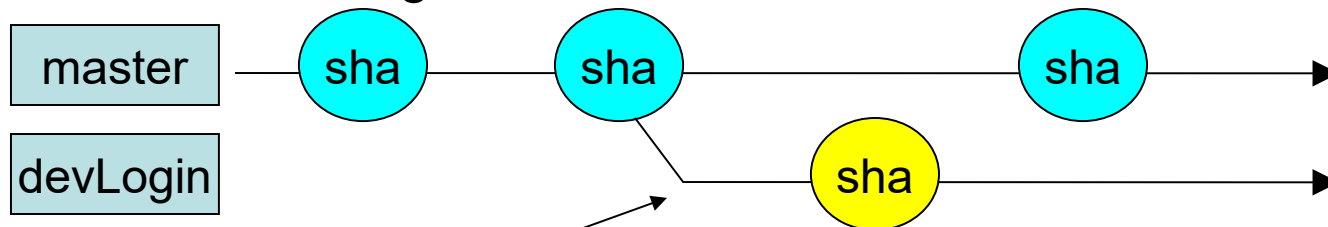
- Gestion de conflit : un même fichier peut être modifié sur GitHub (origin) et en Local (master) de façon contradictoire.
- Un test :
 - Sur GitHub éditer code.txt, modifier une ligne et ajouter une ligne
 - Faire un git commit
 - En local éditer code.txt et faire les mêmes modifications avec du texte différent. Sauvegarder le fichier
 - `$git commit -a -m "modif locale de code.txt"`
 - `$git pull origin main` retourne un conflit sur code.txt et n'effectue pas le pull
 - `$git diff` montre les conflits
 - Éditer le fichier code.txt . Il contient le résultat d'un diff. Garder les lignes souhaitées. Puis effectuer un commit.

- La notion de Branche est un aspect essentiel dans Git
- La branche par défaut est 'master'



 Symbolise un 'git commit'

- Les branches permettent de dériver le chemin principal 'master' en autant de chemins parallèles, souvent le temps d'un développement
Ex: branche 'devLogin'



`git branch devLogin`

- `$git branch devLogin`
`$git branch`
 - crée la Branche
 - montre la liste des Branches
 - * sur la Branche active (HEAD)
- Chaque chemin parallèle – Branche- contient au départ le même contenu que le contenu courant (ici le master)
- Il faut indiquer à Git avec quelle branche on souhaite ensuite travailler.
Git entretient un 'pointeur' vers l'espace courant :HEAD
pour changer le HEAD : checkout

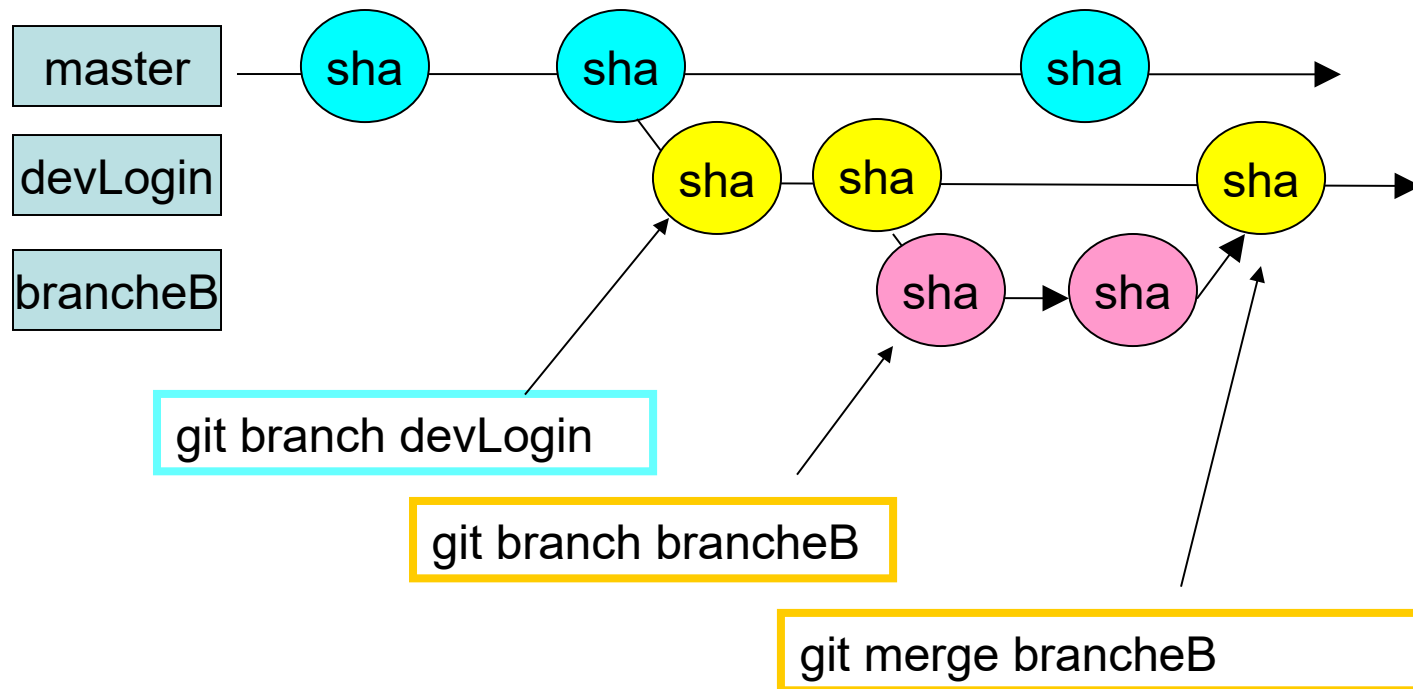
`$git checkout devLogin`
- Les fichiers de ce chemin peuvent évoluer pour satisfaire le dev, des fichiers peuvent être créés

- Editer code.txt et ajouter une ligne spécifique
ex : ligne 5 créée sur branche devLogin
Sauver le fichier
- `$git commit -m "code.txt modifié branche devLogin"`
- En ce point le contenu de code.txt est différent sur master et devLogin. Pour s'en assurer :
 - `$git checkout main` HEAD sur master
 - Consulter code.txt Le contenu est celui de master
 - `$git checkout devLogin` HEAD sur devLogin
 - Consulter code.txt Le contenu est celui de devLogin

- Condition de changement de branche : la branche que l'on quitte doit être 'committed' , pour un travail sain
- Cependant, il est possible de faire une sauvegarde intermédiaire sans commit :
 - Éditer code.txt et ajouter une ligne, sauvegarder
 - `$git stash` 'stash' : stock, planque, cachette
 - `$git stash list` liste les planques de la branche courante
 - `$git checkout master` maintenant permis, sans commit
 - `$git chechout devLogin` retour sur devLogin
 - Consulter code.txt. La nelle ligne est absente
 - `$git stash apply` restitue la planque dans devLogin en gardant le stash
 - OU `$git stash pop` restitue sans garder le stash
 - Consulter code.txt. La nelle ligne est présente

- Création et déplacement dans une branche en une opération :
 - `$git checkout -b mabranche` espace identique au dernier commit
- Création d'une branche à partir d'un ancien commit
 - `$git branch mabranche <sha partiel commit>`

- Fusion de branches



- `$git log` pour récupérer le sha souhaité
- `$git branch brancheB <début sha>` création branche
- `$git checkout brancheB` HEAD pointe vers brancheB
- Créer le fichier codeB.txt
- `$git add code*` référence codeB.txt
- `$git commit -m "ajout codeB.txt"`
- `$git checkout devLogin`
- `$git merge brancheB` devLogin récupère codeB.txt
- En cas de conflit entre des modifications concurrentes dans les deux branches, le/les fichiers contiennent les différences.
Ne garder que le nécessaire puis commit.

- Qui a fait une modification et laquelle ?
 - `$git blame <nomfichier>` fournit les sha dates et personnes
 - `$git show <debut sha>` montre les modifs associées au commit
- Comment ignorer des fichiers dans un repository ?

Certains fichiers propres à la sécurité **ne doivent pas** être gérés.

 - Créer le fichier `security.txt`. On veut l'exclure du repository
 - Créer le fichier `.gitignore` avec la ligne `sec*.*` : exclure tout fichier `sec*`
 - `$git add .gitignore`
 - `$git commit -m "ajout .gitignore"`

- Pousser une branche vers le compte GitHub
 - `$git push origin devLogin`

Le pull request . Permet à un développeur d'indiquer que son travail est disponible pour être intégré dans le projet

<https://www.freecodecamp.org/news/how-to-make-your-first-pull-request-on-github-3/>

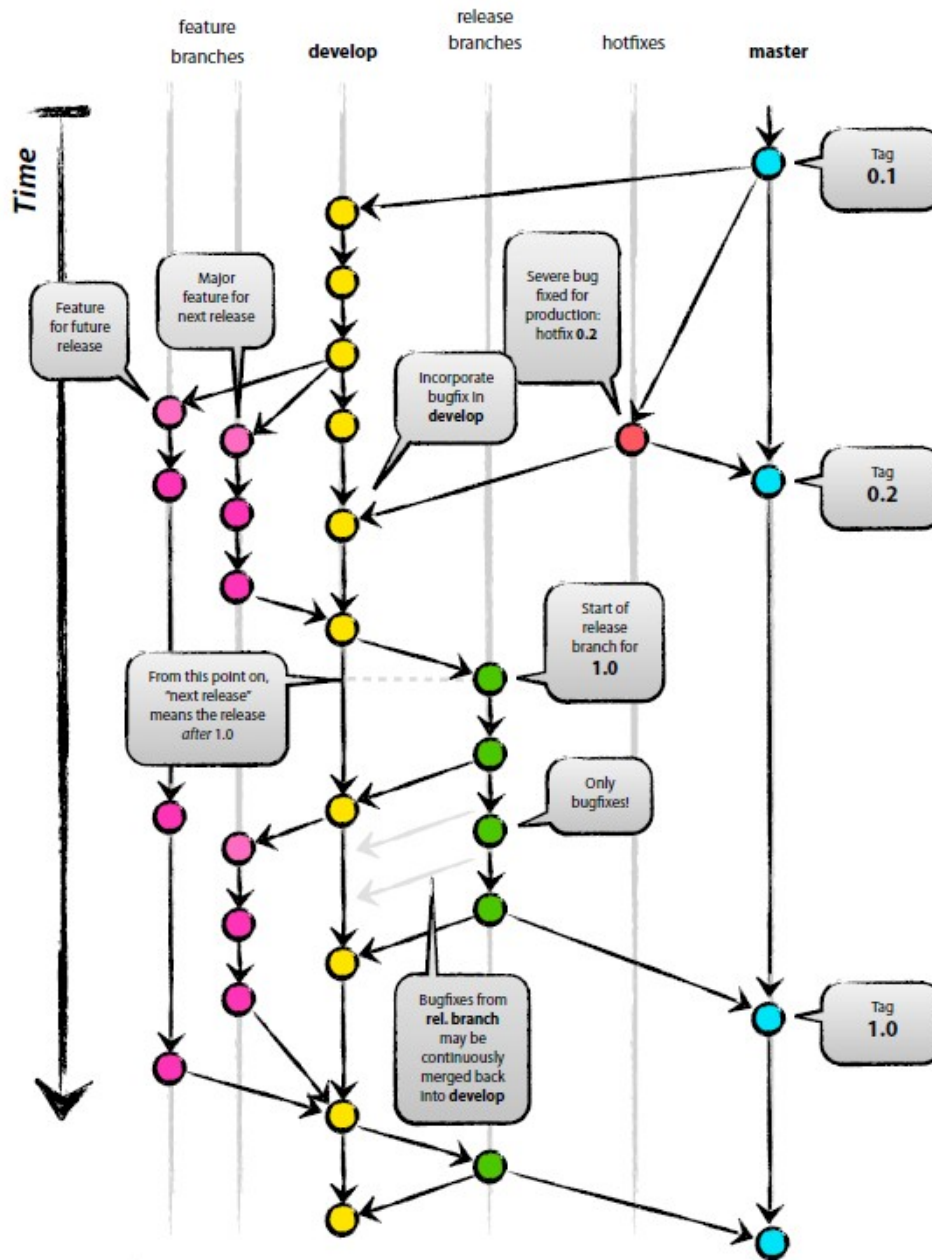
- Autre aspect : le tag – étiquetage d'un commit – fournir un no de version
 - `>git tag -a v10.5 -m "label" <début sha>`
 - `>git tag` montre la liste des tags
- Recherche de texte dans le projet
 - `>git grep "texte à chercher"`

- La gestion/mouvement des fichiers entre la branche principale et d'autres branches se nomme un Workflow
- Seul ou dans une équipe de développement, il faut en début de projet définir clairement comment Git (éventuellement GitHub) va être utilisé.

Exemple :

- La branche master est la seule utilisée pour produire l'application
- Jamais de modification de code dans la branche master
- Tout développement identifié dans le planning est réalisé dans une branche qui porte le même nom que l'activité Gantt
- Qui est responsable des merge dans Git
- ...
- Voici un exemple pour un gros projet :

<http://nvie.com/posts/a-successful-git-branching-model/>



- TortoiseGit propose les fonctions Git sous forme graphique



- GitKraken propose les mêmes choses ... et d'autres fonctionnalités
- <https://appmus.com/vs/gitkraken-vs-tortoisegit>

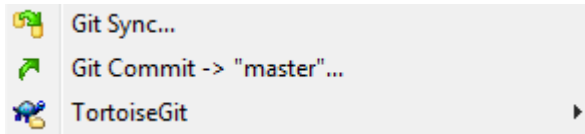
- TortoiseGit propose les fonctions Git sous forme graphique



- Aller sur le site et installer TortoiseGit

<https://tortoisegit.org/download/>

- TortoiseGit est visible depuis l'explorateur Windows avec les nouveaux items de menu



- Pull...
- Fetch...
- Push...
- Diff
- Diff with previous version
- Show log
- Show Reflog
- Browse References
- Daemon
- Revision graph
- Repo-browser
- Check for modifications
- Rebase...
- Stash Save
- Bisect start
- Resolve...
- Revert...
- Clean up...
- Switch/Checkout...
- Merge...
- Create Branch...
- Create Tag...
- Export...
- Add...
- Submodule Add...
- Create Patch Serial...
- Apply Patch Serial...

C:\Users\Daniel\GitDemo\demoVisual2 - Log Messages - TortoiseGit

devLogin From: 23/06/2017 To: 23/06/2017 js, Author Author Email

Graph	Actions	Message	Author	Date
		Working tree changes		
		devLogin autre modif depuis de...	albert	23/06/2017 08:59:58
		program.cs mod dans devLogin	albert	23/06/2017 08:55:56
		master modif program.cs	albert	23/06/2017 08:49:07
		Add project files.	albert	23/06/2017 08:37:16
		Add .gitignore and .gitattributes.	albert	23/06/2017 08:37:10

SHA-1: c27060e1f2655457911925ea90098c9042a744ed

* autre modif depuis devLogin

Path	Extension	Status	Lines added	Lines removed
Program.cs	.cs	Modified	1	0

Showing 5 revision(s), from revision b06941f to revision c27060e - 1 revision(s) selected, 0 file(s) selected

☐ Show Whole Project

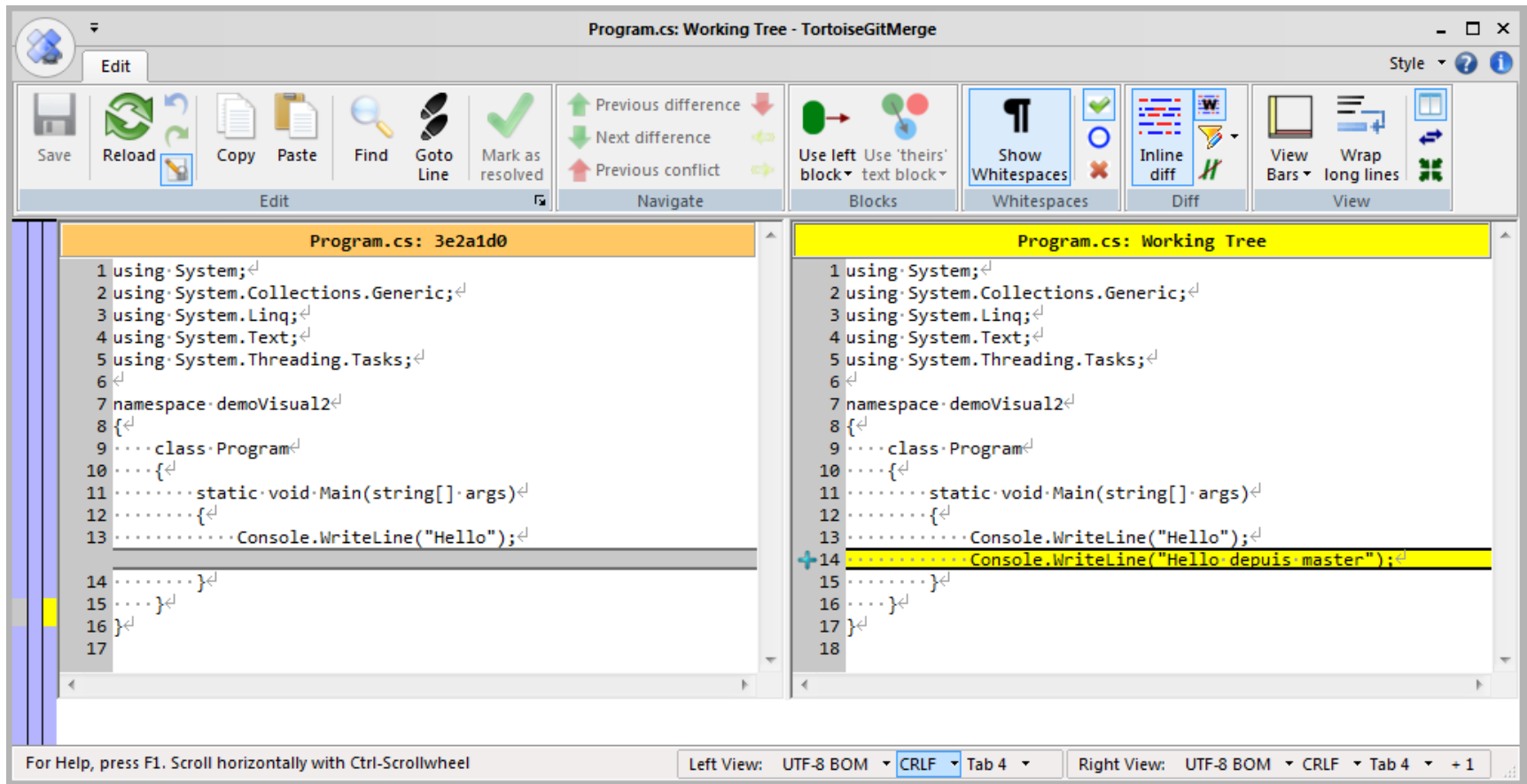
☐ All Branches

Filter paths

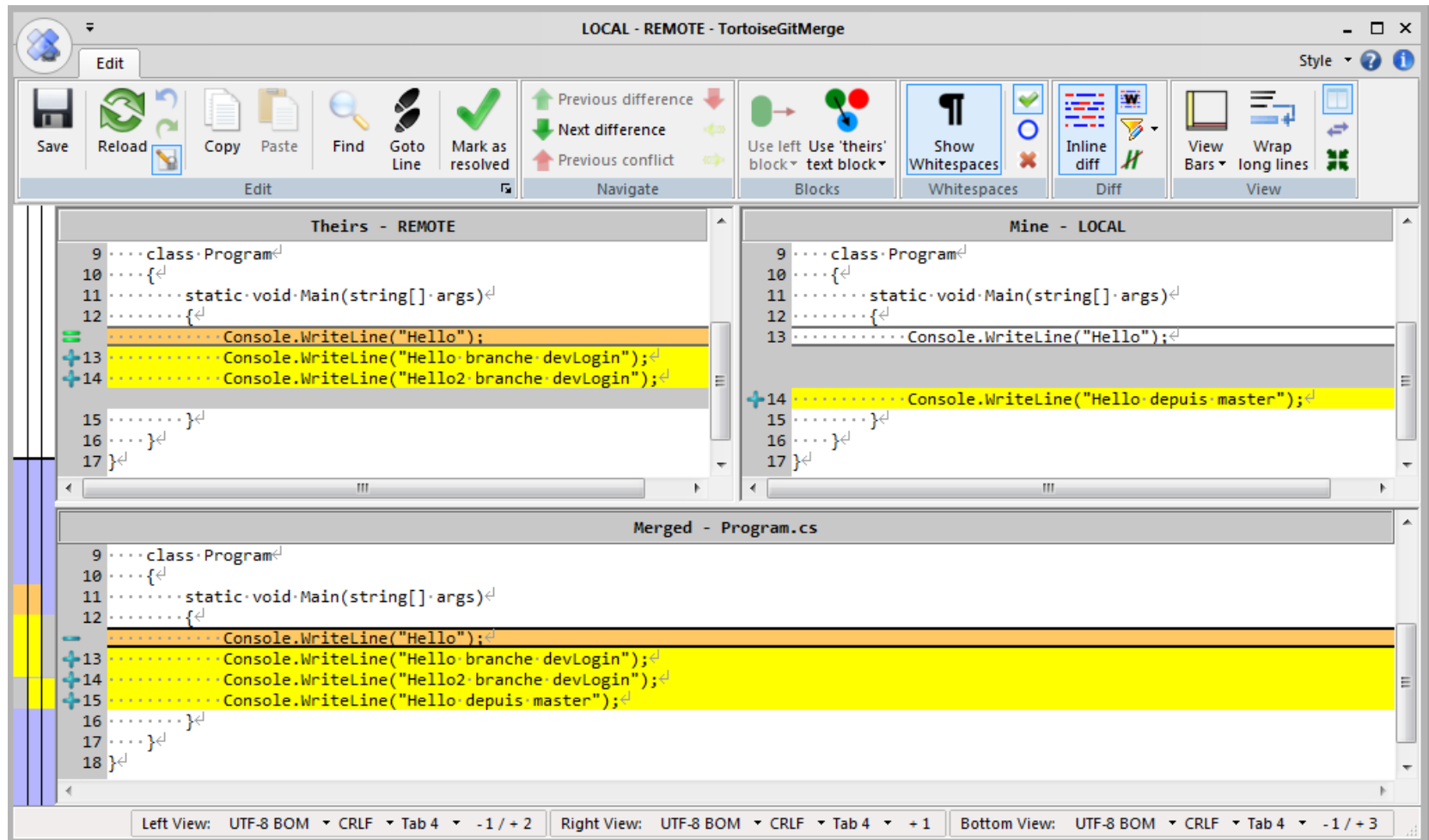
Refresh Statistics Walk Behaviour View

Help OK

- La fenêtre de diff est très complète



- La fenêtre de conflit de merge est très complète

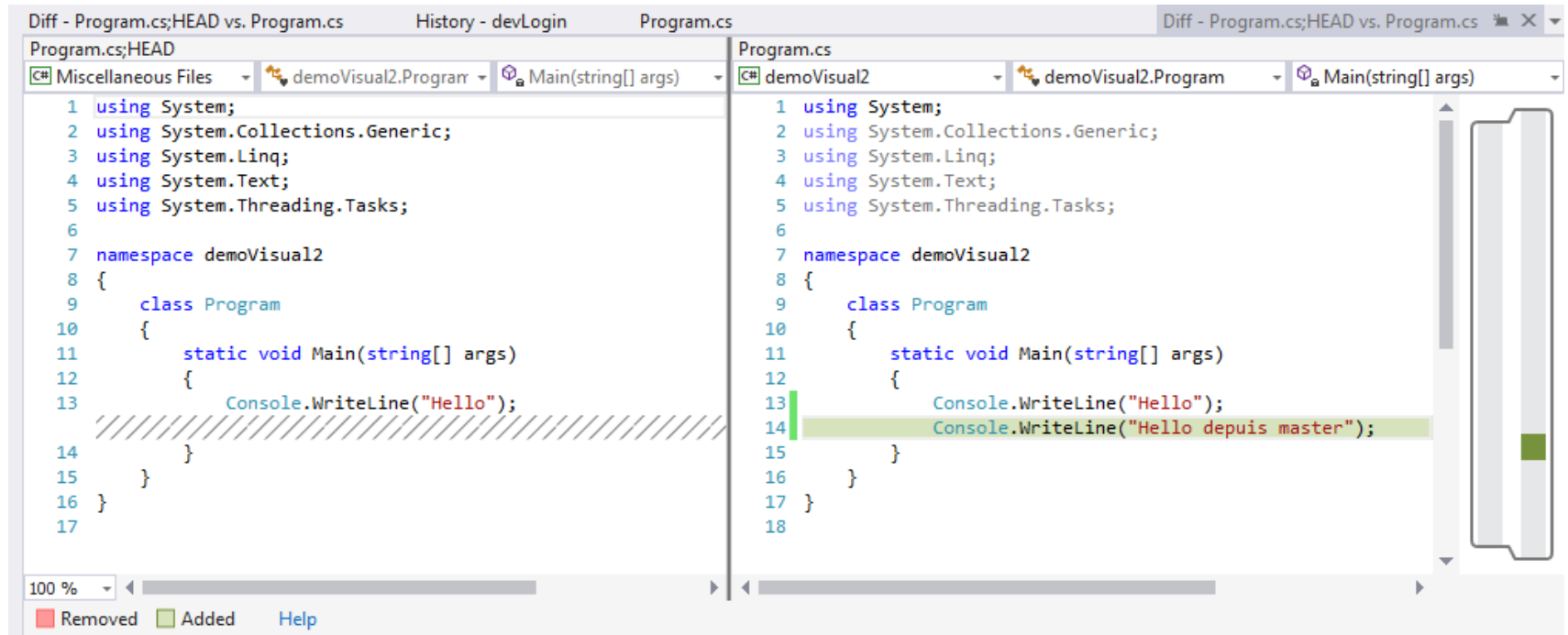


- La plupart des IDE intègrent le versionning avec GIT
- Visual Studio intègre dans sa partie Team Explorer l'interface avec Git
- Depuis Visual il est possible de mettre en version Git un projet existant ou de créer un projet sous version Git

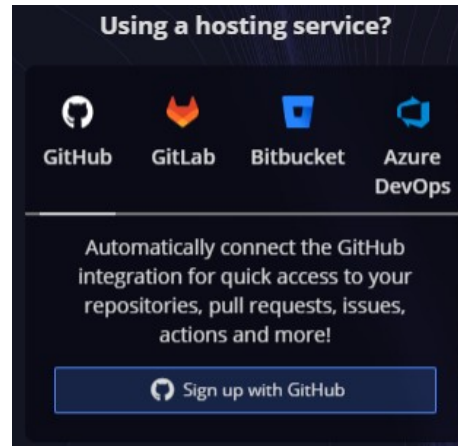


- Ex : créer un projet sous version Git :
 - Au moment de la création, cocher dans la fenêtre New project
- Les menus contextuels proposent des actions propres à Git

- Fenêtre Visual de diff et de gestion de conflit



- Disponible gratuitement sur Windows, Linux, MacOS
- Télécharger sur <https://www.gitkraken.com/download>
- S'authentifier sur GitHub



- Répondre comme suit aux questions posées :

📁 Open an existing repository on your machine

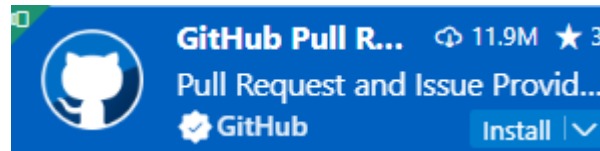
Continue my 7 day trial



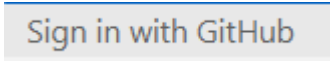

- Vidéo de présentation disponible sur ce lien

https://www.google.com/search?q=youtube+gitkraken+fran%C3%A7ais&client=firefox-b-d&sxsrf=APwXEdTf6C3mUI1EzflHbn0cqcLQcjhE8A%3A1681482419625&ei=s2I5ZO-9JZ6bkdUPjbS08AY&og=youtube+gitkraken+fr&gs_lcp=Cgxnd3Mtd2l6LXNlcnAQAQAgBMqUIIRCgATIFCCEQoAEyBQghEKABOgoIABBHENYEELADog0IABCJBRBHENYEELADogYIABAWEB5KBAhBGABQ3QFYIRJgwz5oAXABeACAAbUBiAHIApIBAzEuMpgBAKABAcgBCMABAQ&sclient=gws-wiz-serp#fpstate=ive&vld=cid:850e722e,vid:daBPgzan_wl

- Regarder avec l'outil le projet demoGit
- Regarder le projet Symfony

- <https://learn.microsoft.com/fr-fr/training/modules/introduction-to-github-visual-studio-code/>
- Installer sur VS Code l'extension GitHub Pull Requests and Issues



- Clic sur 
- Puis sur  is 
- Placer VS Code sur le projet demoGit
- L'icône  donne accès à des commandes git

- Pour l'environnement JAVA, Eclipse offre le même genre de service
=> EGit



- Vive la gestion de version !

