TP 02 - Git & GitHub

Objectifs du TP

- L'utilisation du Git et du GitHub
- L'utilisation Git-ului par Visual Studio Code
- L'utilisation Git par Eclipse
- L'utilisation Git par Visual Studio
- ……

Documentation supplémentaire

- Git en bref
- Git cheatsheet detaliat
- Git reference
- Stanford, Git Magic

Git et GitHub

Git est un système de gestion de code source et de versionnage qui permet le travail sur un projet software. Cela fonctionne généralement sur la ligne de commande; pour Linux et MacOS, il n'y a aucun problème mais les utilisateurs de Windows ne peuvent pas utiliser "cmd" et il y a besoin d'installer "Git Bash".
GitHub est une plateforme en ligne basée sur Git, que les développeurs peuvent utiliser pour stocker et versionner leur code source. Git est l'utilitaire utilisé, et GitHub est le serveur et l'application Web sur lesquels il s'exécute.

Création d'un compte GitHub (si vous n'en avez pas déjà un)

Si vous avez déjà un compte sur GitHub, assurez-vous d'avoir une **photo** avec vous sur votre profil, **votre prénom et votre nom** .
Si vous n'avez pas un compte, allez sur GitHub. Vous pouvez en créer un, en utilisant l'adresse de e-mail universitaire ou l'adresse personnele.
Recommandation Si vous utilisez votre adresse e-mail personnelle, vous pouvez également ajouter votre adresse e-mail universitaire pour bénéficier du pack étudiant GitHub. GitHub offre aux étudiants de nombreux avantages qui sont normalement payés. Si vous souhaitez l'utiliser, vous pouvez suivre les étapes de ce document.

Comment créer un compte?

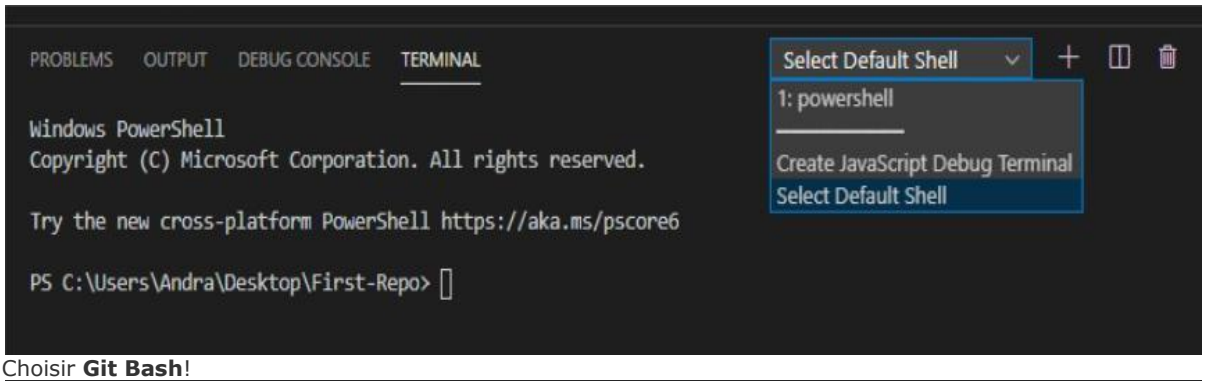
Saisissez un nom d'utilisateur(username), une adresse e-mail et un mot de passe pour votre compte. Pour valider votre compte, accédez à votre e-mail(inbox). Vous trouverez un e-mail expliquant comment valider le compte nouvellement créé. Vérifiez section spam, aussi au cas où vous n'auriez rien reçu dans votre inbox.

Comment rejoindre l'organisation UPB FILS -SdE2?

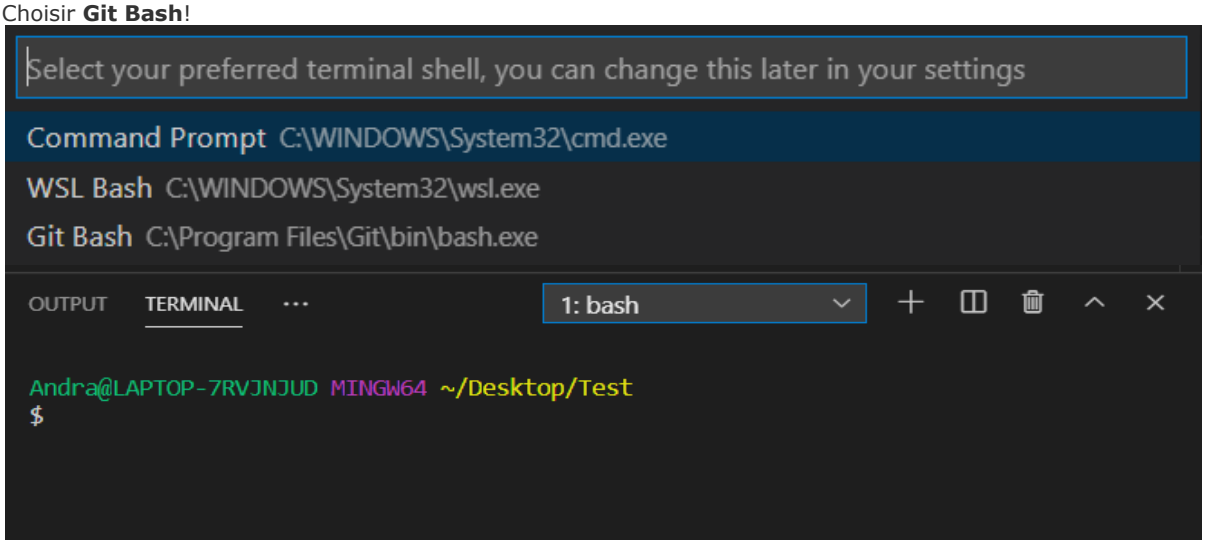
Accéder au lien versUPB-FILS-SdE2. Cliquez sur "Sign in", puis vous devez chercher votre nom d'utilisateur(username), clique dessus et "Accept this assignment". Actualisez la page, puis vous pourrez voir votre repository **Devoir 0**.
Il est obligatoire, pendant le semestre, d'utiliser votre vrai nom, de mettre une photo avec vous sur votre compte GitHub et rejoindre l'organisation UPB-FILS-SdE2. Ainsi, vous ne serez pas identifié pour recevoir des notes pour les devoirs.

Installer Git Bash et l'intégrer dans Visual Studio Code

La première étape consiste à installer Git. Ensuite, ouvrez Visual Studio Code. Dans la barre supérieure, vous voyez **Terminal** , cliquez pour ouvrir un nouveau terminal.
Suivez les images ci-dessous pour ouvrir un terminal Git Bash:



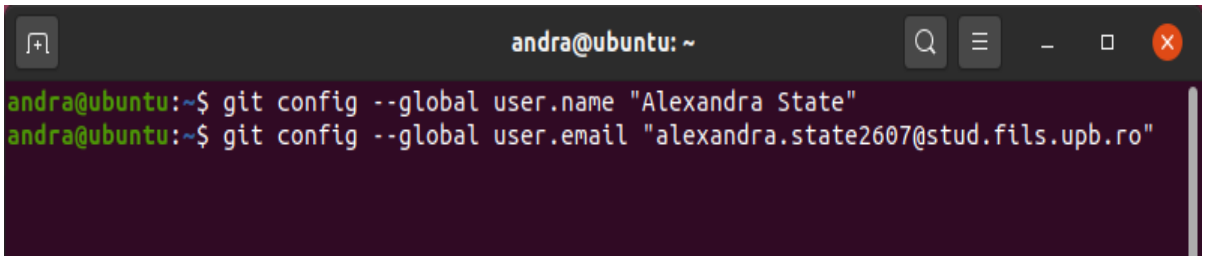
Choisir **Git Bash**!



Réglages de base pour Git

La première étape pour utiliser Git est de configurer votre nom et votre adresse e-mail avec les commandes suivantes:

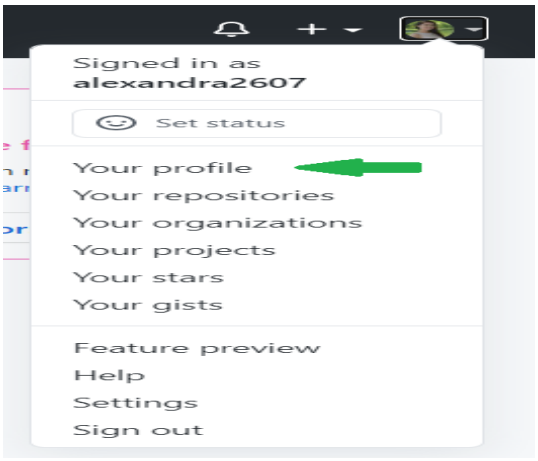
```
user:~$ git config --global user.name "Prenume Nume"

user:~$ git config --global user.email "adresa_de_email@example.com"
```



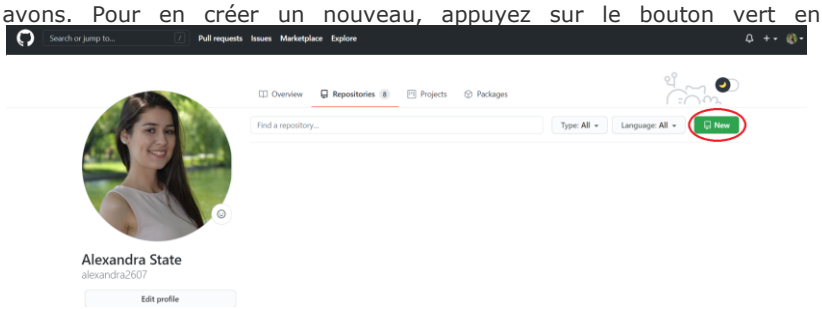
Créer un repository sur GitHub

Nous allons créer un repository sur GitHub,un repository local, après nous les interconnecterons.

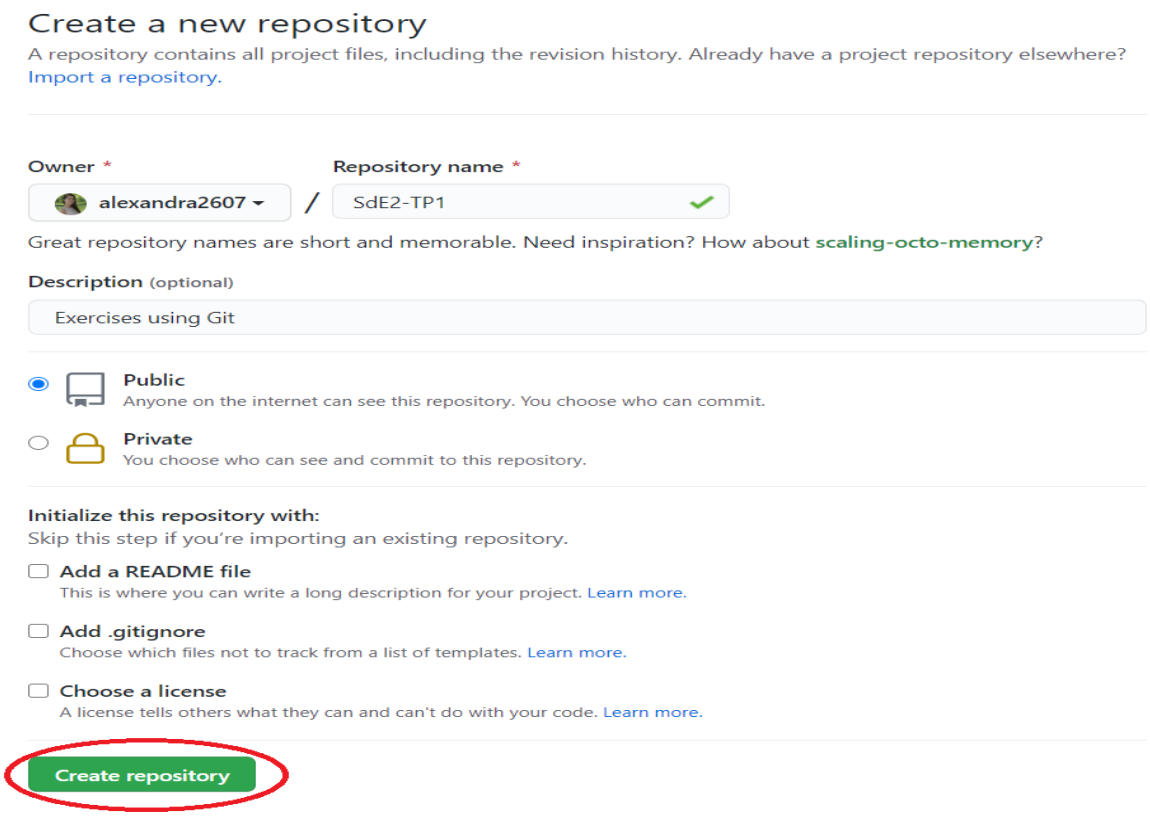
Repository software Le projet est stocké dans un **repository software**. Le repository contient des fichiers de projet: code source, fichiers de configuration. Il est généralement accompagné d'un fichier **README.md** qui contient des informations sur le projet: quel est le but du projet, comment il est compilé, sur quelles plates-formes il s'exécute. Le repository est de deux types: **locale** et **remote**. Ils peuvent être interconnectés et se référer en fait au même projet. Le référentiel local est celui que nous avons sur notre ordinateur, tandis que celui distant est stocké sur un serveur (dans notre cas GitHub). C'est juste une différence de perspective entre les deux, ils ne diffèrent pas techniquement. Habituellement, dans un projet Git / GitHub, il existe un repository central (remote) et plusieurs repositories secondaires (locaux), un pour chaque développeur de l'équipe de projet. Parmi les opérations les plus importantes avec un repository sont:init, fork, clone.



- La première étape consiste à vous connecter à GitHub.
- Ensuite, cliquez sur la flèche dans le menu en haut à droite et voyez quelque chose de similaire à l'image suivante.
- Cliquez sur *Your profile* . C'est ici que nous pouvons voir nos contributions sur GitHub. En haut de l'écran, nous verrons un menu horizontal qui contient 4 options: Overview, Repositories, Projects și Packages.
- On va ppuye sur Repositories. Nous allons maintenant voir la liste complète des dépôts que nous avons. Pour en créer un nouveau, appuyez sur le bouton vert en haut à droite *New*.



- Il est maintenant temps de donner un nom au projet, une brève description de celui-ci et vouz pouvez décider s'il doit être public (visible par tous les utilisateurs) ou privé (visible uniquement par nous et les collaborateurs potentiels du projet). Un formulaire similaire à celui de l'image ci-dessous apparaîtra. Pour ce tutoriel, nous allons créer un référentiel public.
- Cliquez sur *Create repository*.



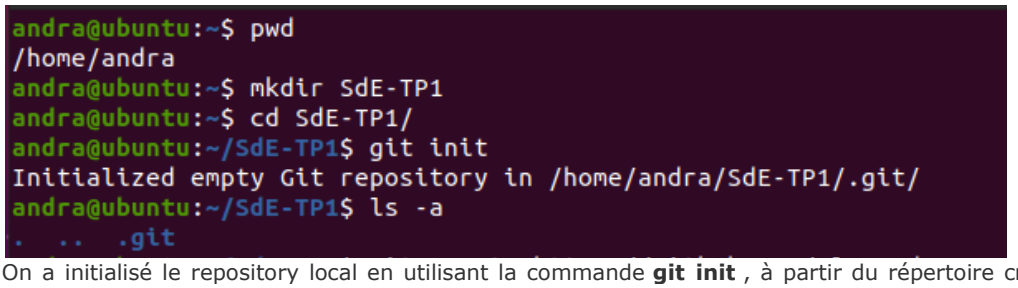
On a maintenant un repository remote créé sur GitHub appelé Sde2-TP1.

Créer un repository local

Dans le répertoire personnel, nous allons créer un répertoire dans lequel on va intialise le repository Git.

Après avoir tapé les commandes suivantes, vous aurez une sortie similaire à celle de l'image ci-dessous.

```
user:~$ mkdir SdE2-TP1

user:~$ cd SdE2-TP1

user:~/SdE2-TP1$ git init

user:~/SdE2-TP1$ ls -a
```



On a initialisé le repository local en utilisant la commande **git init** , à partir du répertoire créé nommé *SdE2-TP1* . En même temps, on a utilisé la commande *ls -a* pour afficher les répertoires cachés, car le répertoire **.Git** est un répertoire caché.

Init

L'opération init est locale et a le rôle d'initialiser un repository local vide. L'initialisation du repository local signifie création, dans le répertoire choisi de l'environnement pour pouvoir travailler sur un projet logiciel versionné Git.Cette opération conduit à la création d'un répertoire nommé .git dans lequel sont des données supplémentaires sur le référentiel, appelées metadate.

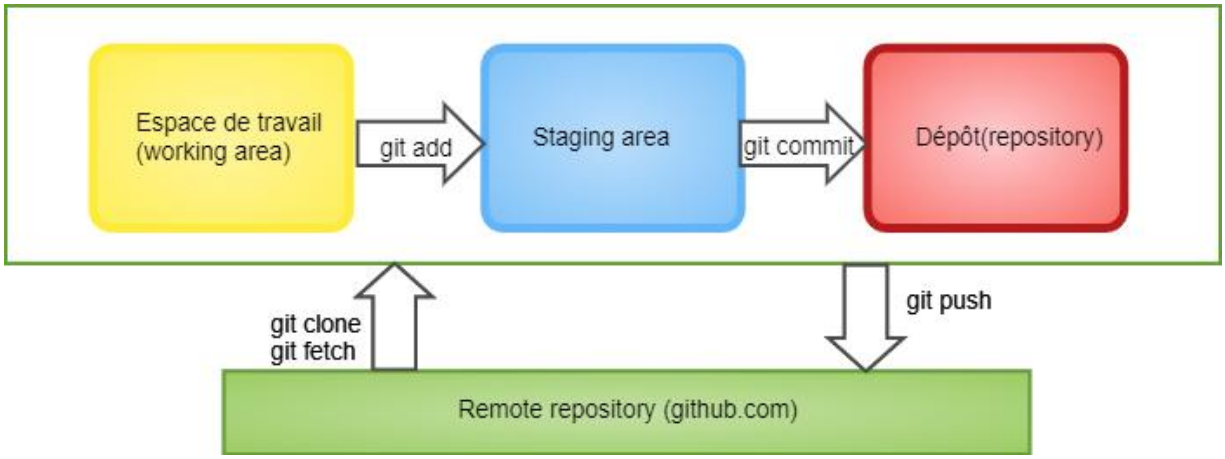
Connecter les deux repositories

Avant, on a créé un repository local et un repository distant, et pour travailler avec eux, on doit les interconnecter, en utilisant la commande:

```
user:~/SdE2-TP1$ git remote add origin https://github.com/{username}/SdE2-TP1.git
```

*{username}*est l' utilisateur sur GitHub, par exemple, selon le nom d'utilisateur dans les exemples précédents, il sera remplacé par *alexandra2607*.
Connecter les deux repositories signifie lr réglage du repository origin, c'est-à-dire le repository remote, au repository local.

La structure générale du Git



Working area est le repository dans lequel vous travaillez, il contient de nombreux fichiers, bien que tous ne puissent pas faire partie du projet final.
Staging area c'est comme un espace incomplet, c'est le lieu où vous pouvez ajouter la version d'un ou plusieurs fichiers que vous souhaitez enregistrer, c'est-à-dire faire partie de votre projet.
Repository c'est tout ce qui se trouve dans la *staging area* et représentera une nouvelle version du projet.

Des commandes de base

git status

Affiche l'état des modifications locales.

git add.

Ajouter des modifications à staging area.

git commit -m "message about the changes made"

Ajouter les modifications déjà du staging are dans le repository local.

git push

"Pousser" les changements de local vers github(remote repository)

git pull

Prenez les changements de remote repository.

git clone

Cloner un repository dans un nouveau répertoire.

Commmit

Travailler sur le projet signifie ajouter et supprimer des fichiers ou modifier des fichiers existants. Ce sont généralement des fichiers texte (humains lisibles), le plus souvent des fichiers de code source. Nous voulons enregistrer ces ajouts et modifications; puis enregistrez à nouveau les autres modifications; etc.

Chaque commit regroupe un ensemble d'ajouts et de modifications effectués par un développeur de projet. En ayant le commits dans le repository on peut gérer le projet plus facilement, c'est à dire:

- revenir à un commit précédent (souvent même le dernier) si les derniers changements "gâchent" le projet
- pour voir qui est l'auteur de certains changements
- pour créer une branche de développement séparée à partir d'un commit précédent, sur laquelle essayer une nouvelle fonctionnalité, sans affecter le reste du projet.

Git s'occupe de maintenir et de gérer l'historique de notre référentiel en conservant la liste des commits pris. Autrement dit, Git conserve un historique des versions du projet.

Enregistrer ces modifications signifie créer un **commit** dans le repository.
Quand on fait commit, il sera conservé dans le repository local du Git, pas dedans le repository remote du Git. Sans mise à jour le repository remote, les autres collègues ne pourront pas voir les changements que nous avons apportés. Nous voulons donc que les modifications apportées localement se retrouvent également remote. C'est-à-dire, les commits du repository local dans le repository remote. On va realiser la publication par l'operation push.

Premiere commit

Les étapes de création d'un commit sont les suivantes:

1. Nous vérifions le référentiel. En d'autres termes, nous vérifions quelles modifications ont été apportées au référentiel depuis le dernier commit
2. Nous ajoutons les fichiers que nous voulons envelopper dans un commit dans la zone de préparation, c'est-à-dire dans la liste des fichiers que Git organise.
3. Nous choisissons un message de validation. Nous créons le commit.
4. Nous publions également le commit sur le référentiel distant.

Exemple: Nous allons créer un fichier README.md que nous publierons dans le repository *SdE2-TP1* sur GitHub.

```
user:~/SdE2-TP1$ touch README.md
```

Vérification de l'état du repository local

Par l'état du repository nous comprenons la forme sous laquelle nous avons amené le projet à travers nos modifications. Cela inclut les fichiers que nous avons créés, modifiés ou supprimés depuis le dernier commit. Nous sommes toujours intéressés par l'état du repository sur lequel nous travaillons.

Pour vérifier l'état du repository, nous utilisons la commande **git status** :

```
user:~/SdE2-TP1$ git status

On branch master



No commits yet



Untracked files:
```

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

        README.md



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

La première ligne affichée On branch master fait référence à la branch master locale.

La deuxième ligne affichée *No commits yet* nous indique que nous n'avons fait aucun commit jusqu'à présent, c'est-à-dire que nous avons commencé à partir d'un repository vide.
Dans la dernière partie de la sortie se trouve une liste de fichiers non suivis, c'est-à-dire la liste des fichiers que Git considère comme nouvellement ajoutés au repository actuel, mais qu'il ne surveille pas encore. Cela signifie que, pour le moment, les modifications que nous apportons à ces fichiers ne seront pas suivies par Git. Dans notre cas, le fichier non suivi est README.md.

Ajouter des modifications en staging area

Un commit contiendra une liste de changements: fichiers ajoutés, fichiers supprimés, contenu modifié. Une étape intermédiaire dans la création d'un commit consiste à préparer les changements qui feront partie du commit. Cette étape de préparation consiste à ajouter (add) ces modifications à l'espace de travail Git, appelé la **staging area** .
Dans notre cas, nous voulons ajouter le fichier README.md nous utiliserons la commande *git add* * , qui chargera toutes les modifications.

```
user:~/SdE2-TP1$ git add *

user:~/SdE2-TP1$ git status

On branch master



No commits yet



Changes to be committed:

  (use "git rm --cached <file>..." to unstage)

        new file:   README.md
```

Les 2 premiers messages affichés sont inchangés. La partie intéressante apparaît dans la dernière partie de la sortie. On voit que le message est devenu des changements à valider. Cela signifie que Git suit maintenant les nouvelles modifications et attend que les modifications soient ajoutées à un commit.

La création du commit local

Maintenant, nous voulons que les changements ci-dessus atteignent dans le repository. Pour cela, nous créons un commit à l'aide de la commande *git commit*:

```
user:~/SdE2-TP1$ git commit -m "Add README file"

[master (root-commit) b051b88] Add README file

 1 file changed, 0 insertions(+), 0 deletions(-)

 create mode 100644 README.md

user:~/SdE2-TP1$ git status

On branch master
```

```
nothing to commit, working tree clean
```

On a mis la description *Add README file* au commit, comme le message pour le commit. Ces doivent être ponctuel et facile à comprendre.

Créer un nouveau commit

Créez un fichier *example.py*, suivez les étapes précédentes et mettez "Add exemples.py" comme message du commit.
Pour voir l'historique des commits, utilisez la commande **git log**.

La publication des commits dans le repository remote

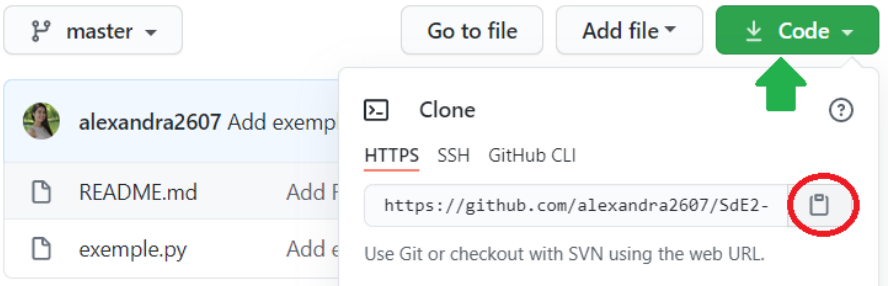
Jusqu'à présent, nous avons apporté des modifications locales, si nous entrons dans la page GitHub dans le référentiel "SdE2-TP1", nous ne pouvons pas voir les modifications, pour cela la commande **git push** est obligatoire.

```
user:~/SdE2-TP1$ git push origin master

Username for 'https://github.com': alexandra2607

Password for 'https://alexandra2607@github.com':

Enumerating objects: 3, done.

Counting objects: 100% (3/3), done.

Writing objects: 100% (3/3), 230 bytes | 230.00 KiB/s, done.

Total 3 (delta 0), reused 0 (delta 0)

To https://github.com/alexandra2607/SdE2-TP1.git

 * [new branch]      master -> master
```

Maintenant, si vous entrez dans GitHub, vous pouvez voir les changements et le nombre de commits dans le repository "SdE2-TP1".

L'utilisation du Git en Visual Studio Code

Si vous n'avez pas encore installé Visual Studio Code, vous pouvez l'installer ici.
Pour apporter en Visual Studio Code, le repository celui que on a créé dans la première partie du TP, nous utiliserons la commande **git clone** suivie par le lien vers repository.



Créez un folder avec le nom *SdE2-TP1-vsc* et ouvrez-le in Visual Studio Code. Ensuite, nous utiliserons le terminal de Visual Studio Code, si vous avez des difficultés à l'utiliser, accédez le lien.
Tappez la commande:

```
git clone https://github.com/{username}/SdE2-TP1.git
```

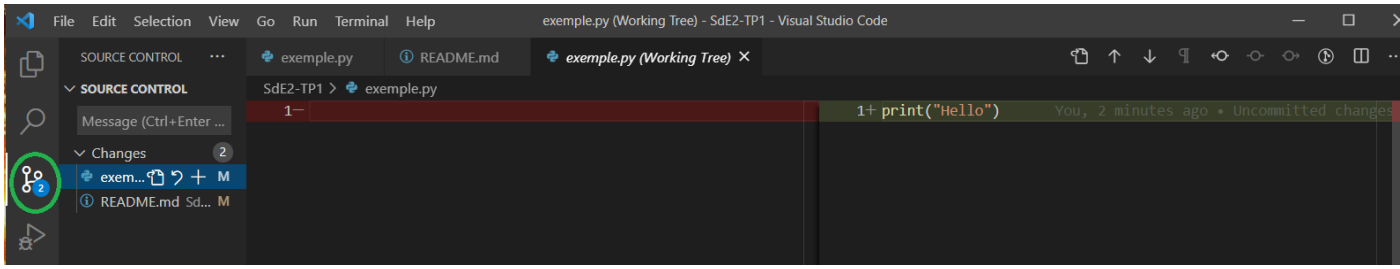
Si c'est la première fois que vous l'utilisez Git en Visual Studio Code immédiatement après cette commande, vous devez suivre les étapes ci-dessous:
Cliquer sur "Sign in with your browser":

Dans le navigateur, cliquez sur *Authorize GitCredentialManager*, et ensuite vous pouvez revenir à VSCode. Nous modifierons le contenu des deux fichiers puis nous avons les mettre sur GitHub.

Écrivez dans le fichier README.md le texte "# SdE2-TP1", et dans le examples.py:

```
print("Hello")
```

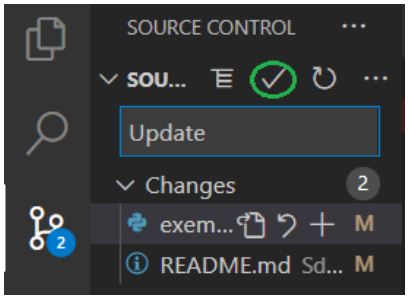
Après les modifications, vous pouvez voir que sur le côté gauche de l'icône de "Source Control" apparaît le numéro 2, qui représente le nombre de modifications. Si vous cliquez dessus, vous verrez une liste des changements, appuyez sur l'un d'eux et VSCode affichera deux versions du fichier modifié et l'original.



Pour enregistrer la nouvelle version du projet et la mettre sur GitHub, nous devons faire commit, puis push. Nous pouvons utiliser les commandes apprises dans la première partie, mais VSCode est livré avec le gestionnaire de contrôle de source Git.

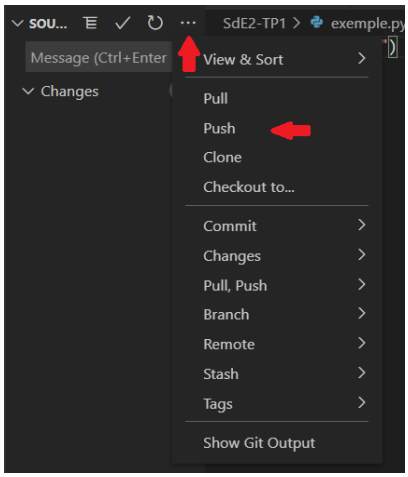
Commit

Dans cet exemple, on a choisi le message de validation "Update", qui est écrit dans le champ de texte, comme le montre l'image ci-dessous. Cliquez ensuite sur l'icône encerclée ci-dessous pour replanifier le commit.



Push

Pour faire "push" nous cliquons sur "More actions"", les 3 points dans le coin, puis un onglet s'ouvre où nous pouvons voir plus de commandes. Nous cliquons sur "push", et les modifications ont été téléchargées sur GitHub, entrez le site dans votre référentiel et vous verrez les modifications.



Modifier les fichiers en ligne

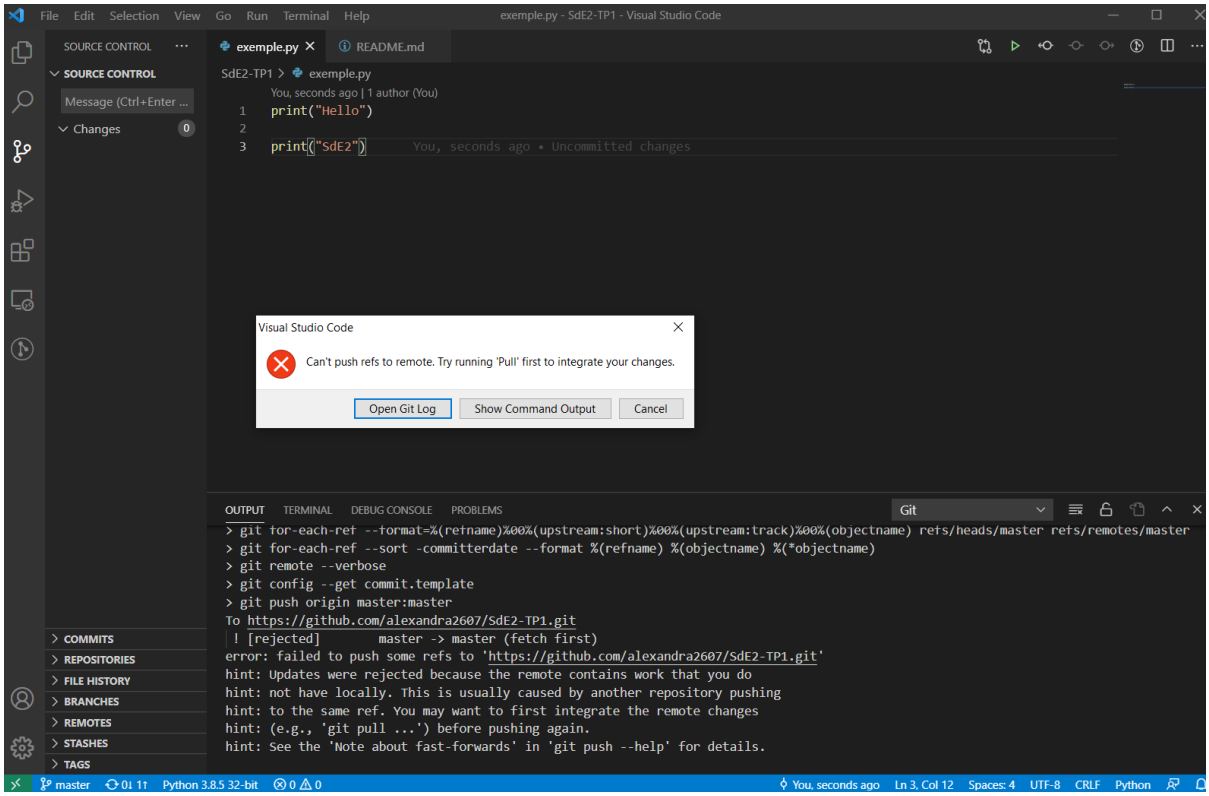
GitHub offre la possibilité de modifier des fichiers directement à partir du repository. Dans cet exemple, nous allons modifier le fichier example.py en mettant le code:

```
print("Hello SdE2")
```

Faites scroll vers le bas de la page et vous verrez un bouton “Commit changes”, appuyez dessus pour enregistrer la nouvelle version.

Pour plus d'informations, lisez ici.
Pour éviter les erreurs, après chaque modification apportée sur la page GitHub, il faut faire **git pull** dans le repository local.

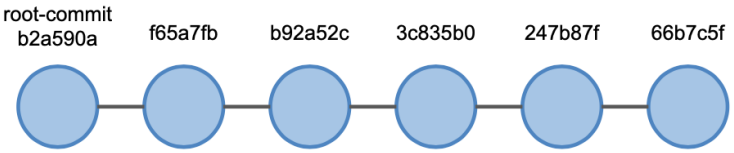
Entrez VSCode et vous remarquerez que le fichier example.py n'a pas de nouvelles modifications. Si nous modifions le fichier sans le mettre à jour localement après la validation, le push avec les nouvelles modifications sera rejeté.



Extra: Branch

Les branches sont utilisées lorsqu'un développeur veut travailler sur une nouvelle fonctionnalité sur laquelle il peut faire de nouveaux commits, ce qui peut déstabiliser le projet. Plus tard, si la fonctionnalité est utile, elle sera ajoutée au projet en unifiant cette branche *(merge)* ; sinon la branche sera supprimée.
Un repository Git a un branch principal de developper, qui s' appelle master. La branche master est la branche par défaut avec laquelle nous travaillons, dans laquelle nous ajoutons des commits et dans laquelle nous voyons l'historique des commits.

Pour voir l'historique des commits on utilise la commande **git log**.
Exemple de représentation graphique des commits.



Root-commit-ul est le premier commit de l'histoire. Tous les autres commits suivent.
En ce moment, sur notre repository, vous n'avez qu'une seule branche - master. Ensuite, nous travaillerons sur notre projet *SdE2-TP1* .
Dans la section suivante, nous ajouterons un fichier .gitignore au projet.

Généralement, nous ajoutons le fichier .gitignore à un projet sur la branche master.
Le fichier .gitignore sera ajouté au repository sur une autre branche que nous appelons add-gitignore pour s'habituer à utiliser des branches.

Ajouter le fichier .gitignore au repository

Dans cette section, nous apporterons des modifications à une nouvelle branche, pas à la branche principale.

Dans un projet Git, nous écrivons les noms de fichiers et les répertoires dans le fichier .gitignore. Ce sont des fichiers et des répertoires dans le répertoire du projet que nous ne voulons pas ajouter au repository.
Par exemple, nous ne mettons pas de fichiers objets et de fichiers exécutables dans le repository car ce sont des fichiers générés pour un certain type de système. Avec le code source, nous pouvons générer des objets et des fichiers exécutables sur notre système.

Nous vérifions la branche sur laquelle nous nous trouvons en utilisant la commande git branch:

```
user:~/SdE2-TP1$ git branch

*  master
```

Creăm un nou branch folosind comanda:

```
user:~/SdE2-TP1$ git branch add-gitignore

user:~/SdE2-TP1$ git branch

  add-gitignore

* master
```

Maintenant, nous avons 2 branches ajoutent gitignore et master et nous sommes sur la branche master.

On accède à la branche add-gitignore en utilisant la commande **git checkout**:

```
user:~/SdE2-TP1$ git checkout add-gitignore

Switched to branch 'add-gitignore'
```

```
user:~/SdE2-TP1$ git branch

* add-gitignore

  master
```

À ce stade, la branche add-gitignore n'est pas différente de la branche master. Lorsque nous apportons des modifications (sous forme de commits), les deux branches divergeront.

Nous ne voulons pas avoir de fichiers objets dans le référentiel, nous allons donc configurer Git pour ignorer ces fichiers. Nous faisons cela en ajoutant la chaîne * .o au fichier .gitignore:

```
user:~/SdE2-TP1$ echo "*.o" > .gitignore

andra@ubuntu:~/SdE2-TP1$ ls -a

.  ..  exemple.py  .git  .gitignore  README.md

user:~/SdE2-TP1$ git status

On branch add-gitignore

Untracked files:

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

          .gitignore
```

Nous créons un commit avec ce changement (git add et git commit) et le publions (git push):

```
user:~/SdE2-TP1$ git add .gitignore

andra@ubuntu:~/SdE2-TP1$ git status

On branch add-gitignore

Changes to be committed:

  (use "git restore --staged <file>..." to unstage)

          new file:   .gitignore



user:~/SdE2-TP1$ git commit -m "Add .gitignore file"

[add-gitignore a34fa81] Add .gitignore file

 1 file changed, 1 insertion(+)

 create mode 100644 .gitignore

user:~/SdE2-TP1$ git push origin add-gitignore

Username for 'https://github.com': alexandra2607

Password for 'https://alexandra2607@github.com':

Enumerating objects: 4, done.

Counting objects: 100% (4/4), done.

Compressing objects: 100% (2/2), done.
```

```
Writing objects: 100% (3/3), 332 bytes | 332.00 KiB/s, done.

Total 3 (delta 0), reused 0 (delta 0)

remote:

remote: Create a pull request for 'add-gitignore' on GitHub by visiting:

remote:      https://github.com/alexandra2607/SdE2-TP1/pull/new/add-gitignore

remote:

To https://github.com/alexandra2607/SdE2-TP1.git

 * [new branch]      add-gitignore -> add-gitignore
```

On a publié le commit dans le repository distant sur la branche add-gitignore. Avec la publication du commit sur GitHub, la branche add-gitignore a été créée dans le repository remote.

L'opération merge entre une branche secondaire et master

Dans la section précédente ci-dessus, on a créé un commit sur la branche add-gitignore.

BAD PRACTICE: apporter des modifications directement sur la branche master.

Nous voulons que la modification apportée par nous, dans ce cas la création d'un fichier gitignore, se trouve sur la branche master. Nous faisons cela via l'opération **merge** , une opération qui unit deux branches: c'est-à-dire qu'il apporte le contenu d'une branche à une autre branche, dans notre cas de add-gitignore à master. Pour faire cela, nous devons être sur la branche master, c'est-à-dire sur la branche dans laquelle nous voulons intégrer les changements.
On va changer sur la branche master local en utilisant la commande **git checkout**, puis pour synchroniser avec le master remote:

```
user:~/SdE2-TP1$ git checkout master

Switched to branch 'master'

Your branch is up to date with 'origin/master'.

user:~/SdE2-TP1$ git pull origin master

From https://github.com/alexandra2607/SdE2-TP1

 * branch            master     -> FETCH_HEAD

Already up to date.
```

La branche master locale est synchronisée avec la branche distante. Nous intégrons la branche add-gitignore dans la branche master à l'aide de la commande **git merge** :

```
user:~/SdE2-TP1$ git merge add-gitignore

Updating 0ee4397..a34fa81

Fast-forward

 .gitignore | 1 +

 1 file changed, 1 insertion(+)

 create mode 100644 .gitignore

user:~/SdE2-TP1$ git status

On branch master
```

```
Your branch is ahead of 'origin/master' by 1 commit.

  (use "git push" to publish your local commits)



nothing to commit, working tree clean
```

Nous avons maintenant un commit dans le référentiel local en plus du référentiel d'origine (Your branch is ahead of 'origin/master' by 1 commit), on une commit qui n'est pas poublie encore. Nous vérifions l'historique des validations pour le voir à l'aide de la commande **git log** . Ensuite, nous devons le publier dans le référentiel d'origine à l'aide de la commande **git push origin master** .
Si nous entrons dans la page GitHub dans notre référentiel, nous pouvons voir que le fichier .gitgnore a été ajouté au master.

Supprimer un branch

Dans cette section, nous supprimerons la branche add-gitignore du repositorylocal et de l'interface GitHub, en utilisant la commande **git branch -d** , puis nous ferons push.

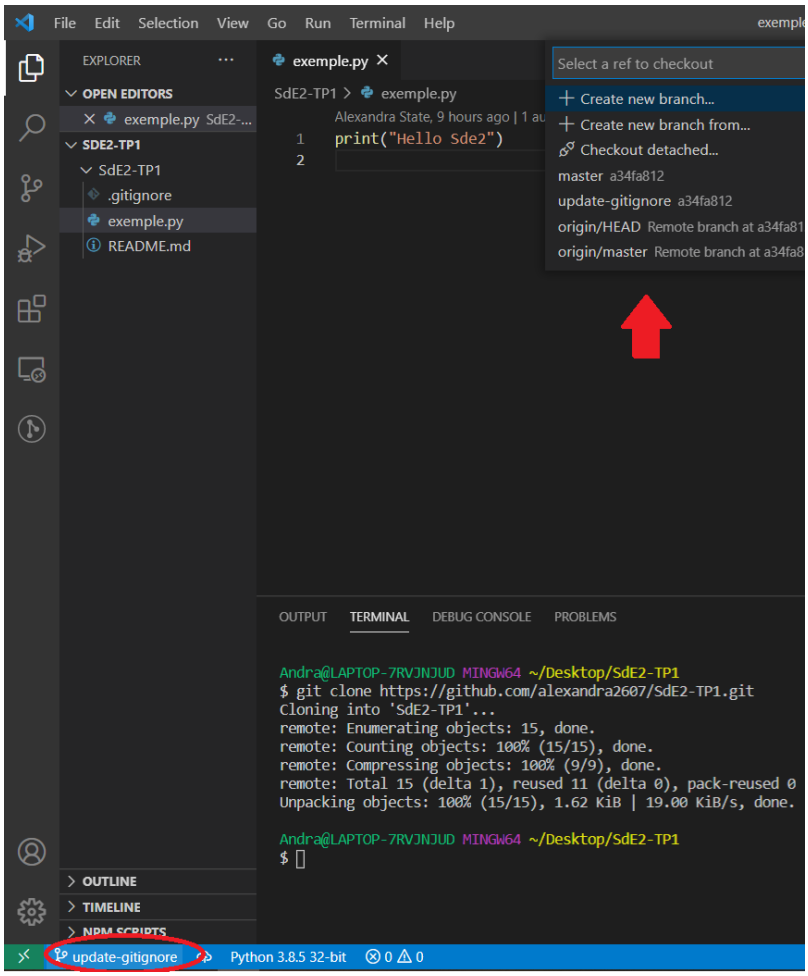
```
user:~/SdE2-TP1$ git branch

  add-gitignore

* master

user:~/SdE2-TP1$ git branch -d add-gitignore

Deleted branch add-gitignore (was a34fa81).

user:~/SdE2-TP1$ git branch

* master

user:~/SdE2-TP1$ git push origin --delete add-gitignore

Username for 'https://github.com': alexandra2607

Password for 'https://alexandra2607@github.com':

To https://github.com/alexandra2607/SdE2-TP1.git

 - [deleted]         add-gitignore

user:~/SdE2-TP1$
```

Branch en Visual Studio Code

Dans cette sous-section, nous reprendrons les étapes présentées dans les sous-sections précédentes. Nous continuerons à travailler sur les brnches. Nous ajouterons une nouvelle ligne au fichier .gitignore, créerons un commit avec cette modification et intégrerons les modifications de la branche secondaire dans la branche master via l'opération merge.

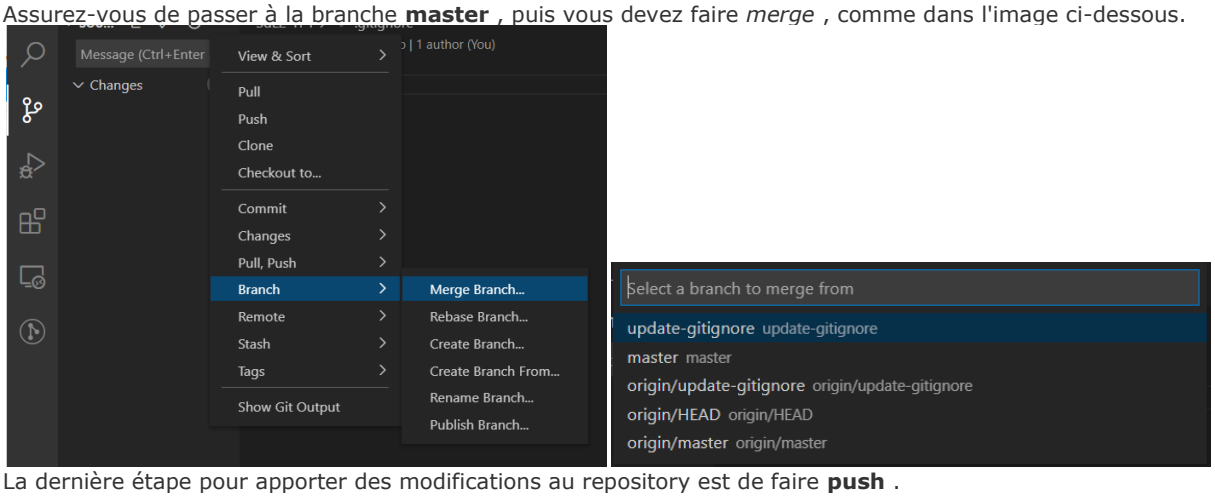
Dans **VSCode** , nous pouvons voir la **branche que nous sommes** dans la barre inférieure, sur le côté gauche où se trouve l'icône de contrôle de source.
Une nouvelle branche est créée en cliquant sur le nom de la branche actuelle, puis VSCode nous donne la possibilité d'en créer une nouvelle, il ne reste plus qu'à entrer le nom.

Nous créons une branche appelée update-gitiginore et VSCode nous y déplace automatiquement. Nous pouvons utiliser le terminal et la commande **git checkout** pour passer d'une branche à une autre ou cliquer sur le nom de la branche et sélectionner la nouvelle.



Nous sommes sur la branche update-gitignore et ajoutons la ligne: build / au fichier .gitignore. Ensuite, nous faisons commit comme on a fait dans la section précédente Commit in VSCode. Pour voir les changements sur la nouvelle branche, il est obligatoire de faire **push** .
On revient à la branche master en cliquant simplement sur le nom de la branche courante, puis en cliquant sur "master". Ensuite, nous effectuons l'opération entre la branche update-gitignore et le master.

Assurez-vous de passer à la branche **master** , puis vous devez faire *merge* , comme dans l'image ci-dessous.



La dernière étape pour apporter des modifications au repository est de faire **push** .

Exercices

1. Créez un compte GitHub.
2. Rejoignez l'organisation UPB-FILS-SdE2.
3. Créez le repository Sde2-TP1 et suivez les étapes présentées dans le TP pour le modifier.

À la fin du TP vous devez charger sur Moodle une fichier avec:

1. Une photo avec votre profil sur GitHub(nom + photo).
2. Une photo avec l'accès du repository pour le Devoir0.
3. Un fichier texte, qui contient le lien vers le vôtre repository SdE2-TP1.