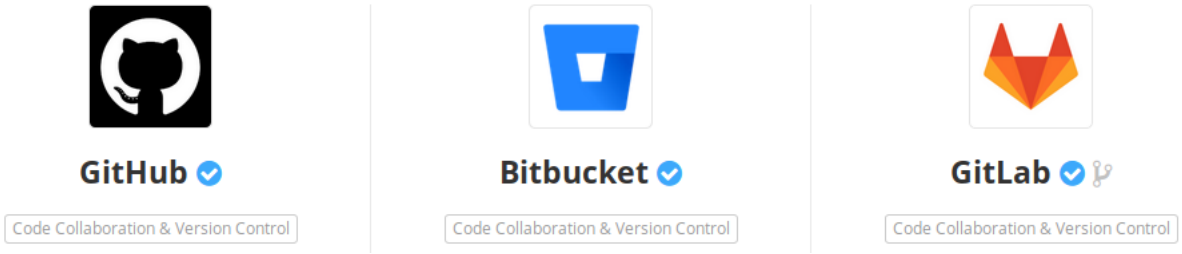
UE1 cons : Consolidation Bases de Données et Programmation Objet.
Versionnage et Tests.

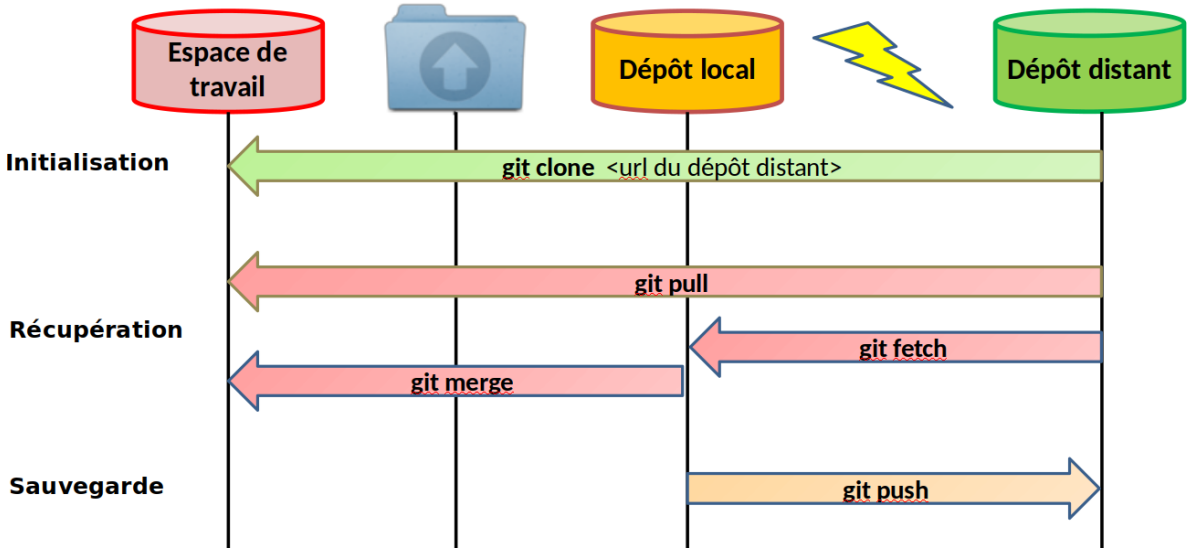
Cours-TD n°2

Introduction à git

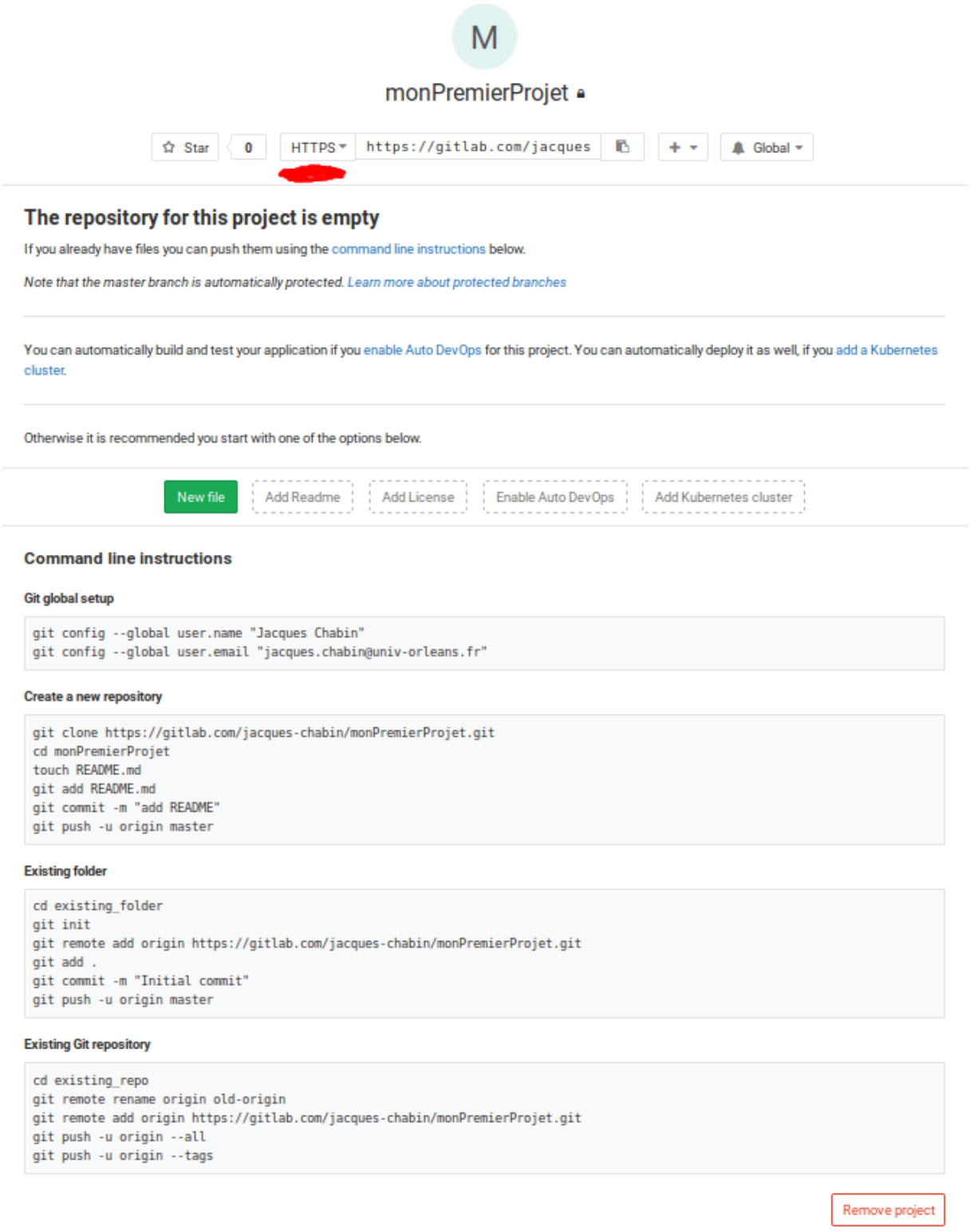
Cette seconde feuille va aborder l'utilisation d'un dépôt git distant.
L'intérêt d'avoir un dépôt distant est de pouvoir le récupérer dans différents environnements mais aussi de pouvoir le partager facilement pour le compléter ou le diffuser.
Il existe plusieurs sites permettant d'héberger et de partager vos projets GIT comme par exemple :



Chacun avec des particularités, gratuit, payant, public, privé, limité en taille etc ... Certaines entreprises ont leur propre serveur Git, l'Université d'Orléans a mis en place un tel serveur. Nous allons utiliser gitlab pour notre dépôt distant. Avant de voir la marche à suivre pour utiliser un dépôt distant, complétons notre schéma général :



1. Créez un compte sur `https://gitlab.com`.
2. Créez un projet, à partir de votre répertoire suivi. Allez dans le menu Projects puis Your projects. Cliquez sur New project (en vert), donnez un nom : `monPremierProjet` puis cliquez sur Create project. (Le projet sera privé il n'y aura pas automatiquement de fichier README etc...). Vous arrivez sur l'écran suivant :



Remplacer SSH par HTTPS juste en dessous du nom de votre projet. Ensuite on vous donne les commandes à faire pour lier votre dépôt distant à votre dépôt local avec trois possibilités :

(a) Vous n'avez encore rien en local : Create a new repository. Vous vous placez à un endroit où vous pouvez créer un répertoire puis :

```
1 git clone https://gitlab.com/jacques-chabin/monPremierProjet
    .git
2 cd monPremierProjet
3 touch README.md
```

```
git add README.md
git commit -m "add README"
git push -u origin master
```

(b) Vous avez un répertoire local qui n'est pas encore suivi (il ne contient pas de répertoire caché `.git` : Existing folder. Vous vous placez le répertoire que vous voulez suivre `cd existingfolder` puis :

```
git init
git remote add origin https://gitlab.com/jacques-chabin/
   monPremierProjet.git
git add .
git commit -m "Initial commit"
git push -u origin master
```

(c) Vous avez un répertoire local qui est suivi (il contient un répertoire caché `.git` : Existing Git repository. Vous vous placez ce répertoire que vous voulez suivre `cd monPremierProjet` puis :

```
git remote rename origin old-origin
git remote add origin https://gitlab.com/jacques-chabin/
   monPremierProjet.git
git push -u origin --all
git push -u origin --tags
```

Nous sommes dans ce dernier cas, nous allons associer notre répertoire suivi à ce dépôt.
`git remote rename origin old-origin` n'est pas nécessaire car le répertoire `monPremierProjet` n'a pas encore été lié à un dépôt distant. Ceci sera utile lorsque votre projet proviendra d'un autre dépôt par exemple.
`git remote add origin https://gitlab.com/jacques-chabin/monPremierProjet.git` cela va créer un lien (un remote) nommé origin (le lien principal) vers le dépôt distant. Vous pouvez voir la création de ce lien avec la commande `git remote -v`. Vous aurez le droit de lecture (fetch) et écriture (push) sur ce lien.
`git push -u origin --all` vous copiez sur le lien distant tout votre dépôt local.
`git push -u origin --tags` vous copiez aussi les tags que vous avez mis sur vos commits.

La commande `git status` va vous dire que tout est ok.

3. Pour éviter de retaper tout le temps son login et son mot de passe on peut faire la commande suivante :

```
git config --global credential.helper 'cache -- timeout=3600
```

Le nombre de secondes ou le login/mot de passe est conservé est limité.

Vous pouvez aussi créer un fichier `.netrc` à la racine de votre `home` contenant les lignes suivantes :

```
machine gitlab.com
login jacques.chabin@univ-orleans.fr
password xxxxxxx
```

login est suivi par l'adresse que vous avez utilisée pour créer votre compte sur gitlab.com, password est suivi de votre mot de passe en clair oups!!! Avec ce fichier vous n'aurez jamais à donner vos identifiant/mot de passe pour accéder au serveur.

4. Comment travailler seul avec un dépôt distant.

   (a) Si c'est la première fois que l'on va travailler avec ce dépôt il faut commencer par le cloner.

   ```
   1 git clone https://gitlab.com/jacques-chabin/monPremierProjet
       .git
   ```

   (b) On récupère la dernière version de dépôt distant dans le répertoire suivi avec la commande `git pull`. Ceci est inutile si l'on vient de cloner le projet.

   (c) On complète le projet en faisant un dernier commit lorsque l'on a fini de travailler. Puis on met à jour le dépôt distant avec la commande `git push`.

   (d) Mise en pratique :

      i. Ajoutez une ligne à rapport.txt. `git status` vous dit que le dépôt distant est à jour avec le dépôt local et que le fichier a été modifié mais pas mis dans l'index.

      ii. Ajoutez rapport.txt à l'index. `git status` vous dit que le dépôt distant est à jour avec le dépôt local et que le fichier rapport.txt est dans l'index mais pas encore dans le dépôt local.

      iii. Ajoutez rapport.txt au dépôt local. (Faites un nouveau commit ou une nouvelle version). `git commit -m"Partie 6 finie"`. `git status` vous dit que le dépôt local sur la branche master est en avance d'un commit par rapport à origin/master (dépôt distant).

      iv. Mettez à jour votre dépôt distant : `git push` (Vous n'avez pas besoin de donner ici la commande complète `git push -u origin master` (tant que vous ne changerez pas avec une commande du type `git push -u xxxx yyyy`, git conservera la dernière option utilisée).

5. Comment travailler à plusieurs avec un même dépôt distant. Il risque d'y avoir des incompatibilités si plusieurs personnes modifient le même fichier en même temps. Une première solution consiste à ce que chaque développeur du projet travail sur des fichiers différents. Une seconde solution consiste à utiliser la notion de branche.

6. On va simuler la présence de plusieurs développeurs en créant deux répertoires (que l'on nommera **Alice** et **Tom** en dehors de votre répertoire suivi **monPremierProjet**.

7. Placez-vous dans votre répertoire **Alice**. Dans ce répertoire nous allons cloner notre dépôt distant. Pour cela tapez la commande :

   ```
   1 git clone https://gitlab.com/jacqu-chabin/monPremierProjet.git
   ```

   en remplaçant l'adresse de mon projet par l'adresse de votre projet. (Vous avez cette adresse facilement en vous connectant sur gitlab.com en cliquant sur votre projet et en remplaçant ssh par https.)

8. Faites la même chose dans le répertoire **Tom**. Vous avez maintenant 3 répertoires distincts qui sont en lien avec le même dépôt distant.

9. Si Tom et Alice travaillent à tour de rôle, en suivant la démarche :

   ```
   1 git pull
   2 /* modification du projet ajout de fonctionnalités etc ... */
   3 git commit
   4 git push
   ```

Il n'y aura pas de conflit et donc pas de soucis.

Mise en pratique :

Alice crée un nouveau fichier **annexe.txt** le versionne et le pousse dans le dépôt distant.

Tom complète le fichier **annexe.txt** en ajoutant une seconde ligne puis en créant une version qu'il pousse sur le dépôt distant. Attention Tom doit récupérer la dernière version du dépôt distant avec `git pull`.

Récupérez cette dernière version dans les répertoires **monPremierProjet** et **Alice/monPremierProjet**.

10. Supposons maintenant que Ton et Alice travaillent en même temps mais sur des fichiers distincts.

    Alice ajoute le fichier **alice.txt** le versionne mais ne le pousse pas sur le dépôt distant (ne fait pas `git push` à la fin.

    Tom ajoute le fichier **tom.txt** le versionne mais ne le pousse pas sur le dépôt distant (ne fait pas `git push` à la fin.

    Alice pousse sa dernière version sur le dépôt distant. Tout se passe bien.

    Tom pousse a dernière version sur le dépôt distant. Le `git push` est refusé car le dépôt distant contient des choses que l'on n'a pas en local, il faut d'abord récupérer ce travail par un `git pull`. Ceci fera une fusion (un merge) en local de la branche distante et de la branche locale en créant une nouvelle version locale. (Il n'y a pas de conflit car les fichiers qui ont été modifiés sont différents.) C'est cette nouvelle version qui peut maintenant être poussée sur le dépôt distant `git push`.

    Récupérez cette dernière version dans tous les répertoires avec `git pull`.

11. Si Alice et Tom travaillent en même temps sur les mêmes fichiers.

    Alice ajoute une ligne au fichier annexe.txt le versionne mais ne le pousse pas sur le dépôt distant (ne fait pas `git push` à la fin.

    Alice ajoute une ligne au fichier annexe.txt le versionne mais ne le pousse pas sur le dépôt distant (ne fait pas `git push` à la fin.

    Alice pousse sa dernière version sur le dépôt distant. Tout se passe bien.

    Dans le répertoire **Tom/monPremierProjet**, Tom pousse la dernière version sur le dépôt distant. Le `git push` est refusé car le dépôt distant contient des choses que l'on n'a pas en local, il faut d'abord récupérer ce travail par un `git pull`. Ici la fusion n'est pas possible car il y a un conflit. Il faut régler le conflit avant de faire cette fusion. Le conflit a été "généré" dans le fichier **annexe.txt**, une solution est d'éditer ce fichier. Ce fichier contient trois parties, les parties qui sont communes, les parties qui sont issues du travail de Tom (sous la bannière `"<<<<<<< HEAD"`) et les parties qui sont issues d'un travail extérieur rapatrié par le pull. On choisit ici ce que l'on garde ou supprime dans ce fichier (on supprime les bannières). On versionne une nouvelle fois et on peut maintenant pousser cette version sur le dépôt distant.

    Récupérez cette dernière version dans tous les répertoires avec `git pull`.

12. Refaites cette même démarche mais en utilisant deux personnes différentes, deux développeurs sur le même projet. Mettez-vous en binôme, un sera Alice l'autre Tom.
    - Alice créé un dépôt vide (blanck project) sur gitlab.
    - Alice invite Tom à partager son projet : `Settings --> Members` Choisir Tom dans les membres de gitlab puis lui affecter un rôle. (`Guest` (soumettre des bug), `Reporter` (lire le projet), `Developer` (écrire dans le projet), `Maintainer` (inviter dans le projet)).

— Alice clone le projet chez elle.
— Tom clone le projet chez lui.
— Alice travaille et versionne son travail.
— Tom récupère le travail d'Alice, développe une nouvelle fonctionnalité et versionne.
— Alice récupère le travail de Tom.
— Etc ... Il ne doit pas y avoir de soucis si vous travaillez l'un après l'autre ou sur des fichiers séparés sinon il faudra gérer les éventuels conflits.

13. Remarque, cette solution n'est pas élégante nous allons passer par la notion de branche. La branche master sera la branche principale de votre projet et chaque nouvelle fonctionnalité sera développée dans une nouvelle branche. Lorsque la fonctionnalité sera terminée, la branche qui la contient sera fusionnée avec la branche principale (master) par **la** personne (chef de projet) qui gérera la branche master.