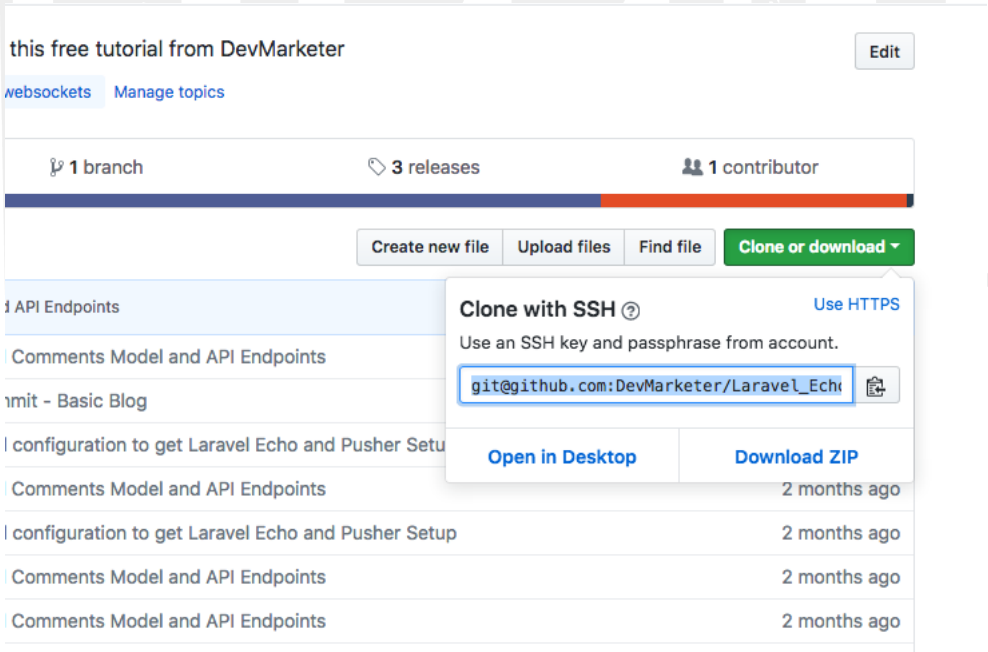
Guide de survie pour Git

Github, qu'est ce que c'est ?

Github est un service en ligne qui permet d'héberger ses repositories de code. Github est un outil gratuit pour héberger du code open source. Pour commencer à utiliser Github il faut créer son compte sur la page page d'acceuil, rentrer son nom d'utilisateur , un mot de passe etc. Une fois cela fait, il faut penser à communiquer son pseudo pour pouvoir accéder au dossier d'eirbot.
Sur Linux il suffit d'avoir un terminal pour faire toutes les commandes git. En revanche si vous êtes sur Windows...tant pis [1]

Récupérer du code d'un autre d'un autre repository

Une fois le compte Github créé, du code d'un autre repository peut être récupéré. Pour cela il faut **cloner** le répertoire, il suffit de cliquer sur "clone" sur Github et de copier le lien en https. (On pourra modifier pour pouvoir utiliser le ssh plus tard).



Une fois le lien copié on tape dans un bash la commande suivante

git clone https ://github.com/eirbot/eirbot2020-1A.git

Cela permet de créér une copie locale du dossier sur notre machine.

1. L'utilisation de logiciel comme Git SCM permettra de réaliser toutes les actions après chacun peut utiliser son logiciel préféré, vive EMACS

Envoyer du code sur Github

Maintenant qu'une copie locale du dossier existe sur la machine, le code (ou n'importe quel fichier) peut être modifié de manière locale. Lorsque le travail terminé il faut synchroniser les modifications effectuées sur le repository local avec le repository distant. L'opération que nous allons décrire permet de valider un ensemble de modifications dans le code pour créer une nouvelle révision, et est communément appelé un **commit**.
Commençons par visualiser l'état des fichiers dans le dépôt avec la commande suivante [2]

git status

Si le fichier n'a jamais été communiqué au dépôt, le résultat devrait être être de la forme

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

      nom_du_fichier
```

Si le fichier était déjà présent dans le dépôt, le résultat devrait être de la forme :

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   nom_du_fichier
```

Une fois que nous connaissons l'état des fichiers il faut choisir ceux que l'on veut envoyer pour cela

git add nom_du_fichier

Une fois les fichiers ajouté nous pouvons **commit** ces derniers via

git commit

Cela va ouvrir un éditeur de texte, il faut ajouter un commentaire. Une fois le commentaire validé la révision a été créé.
Attention ! Les commentaires des commits sont les premières choses que les collaborateurs vont voir, un commit comme "pause dej" n'a aucun intéret [3], les collaborateurs ne peuvent pas comprendre ce que vous avez fait. Il est donc conseillé de commit fichier par fichier en expliquant correctement les modifications réalisées sur chaque fichier.
Comme il s'agit d'une modification des sources, il faut la communiquer au dépôt distant. Le dépôt distant se nomme par défaut origin, et la branche à communiquer master. Communiquer votre modification des sources avec la commande suivante :

git push origin master

2. Cette étape est assez pratique au début pour comprendre où l'on va, on s'en passe rapidement
3. Cela a plutôt tendance à énerver vos collègues et les conduire à détruire chaqu'un de vos commits (oui oui c'est possible)

Récupérer des modifications

Pour récupérer en local les dernières modifications du repository Github, il faut utiliser la commande la commande

git pull origin master

Lorsque l'on réalise un **git pull** on effectue un fetch puis un merge. Dans nos conditions de travail, il est préférable de réaliser un rebase pour que cela soit automatique il faut taper la commande suivante.

git config - - global branch.master.rebase true

Guide avancé pour Git

Gérer un conflit sur du code

Du fait de la présence de multiples dépôts dans lesquels le code peut être modifié, il est possible que la même branche master ait divergé entre deux dépôts (typiquement avec la même branche sur un dépôt distant, comme origin/master). Dans ce cas, une fusion/merge risque d'amener à un conflit. Il convient de ne surtout pas paniquer, cela fait partie des tracas standards du développement logiciel.
Mettons que la dernière fusion/merge ait amené au résultat suivant :

```
Auto-merging nom_du_fichier
CONFLICT (content): Merge conflict in nom_du_fichier
Automatic merge failed; fix conflicts and then commit the result.
```

Il faut bien noter que l'opération de fusion/merge ne s'est pas terminée ("no changes added to commit"). Git vous met dans un état où il est possible de l'assister à terminer cette fusion correctement.
Pour chaque fichier en conflit, appliquer la procédure suivante :

1. Ouvrir le fichier dans un éditeur (dans notre exemple nom_du_fichier).
2. Rechercher dans l'éditeur les blocs de la forme suivante :

   ```
   <<<<<<< HEAD
   Teddy , le programmeur extremiste.
   =======
   Teddy , le programmeur de l'extreme.
   >>>>>>> origin/master
   ```

   Git a modifié de lui-même le fichier pour inclure, aux endroits qui lui semblaient différents, les deux versions. Il est facile de rechercher ces blocs car les marqueurs <<<<<<<, ======= et >>>>>>> ne sont quasiment jamais utilisés.
   La zone délimitée par <<<<<<< et ======= représente la version locale du code (branche master). La zone délimitée par ======= et >>>>>>> représente la version distante du code (branche origin/master). Modifier le code de manière à éliminer tous les marqueurs. Il est ainsi facile de choisir une version des deux versions à garder, ou de faire une sorte de mélange des deux si besoin.
3. A la fin des modifications, ajouter le code au commit (ici le fichier en conflit)

   git add nom_du_fichier
4. Une fois tous les fichiers en conflit gérés, terminer la fusion (un message de commit est généré automatiquement) :

   git commit

Authentification via clé SSH

Si vous voulez utilisez cloner quelque chose des dépôts distants, vous devrez choisir entre un des deux moyens : HTTPS ou SSH. Si vous utilisez HTTPS, vous devrez taper l'accès de votre compte chaque fois pour communiquer avec le dépôt distant, il y a un moyen de contourner le problème, la méthode d'authentification SSH.

Pour modifier son authentification vers ssh il faut tout d'abbord générer une clé privée et une clé publique sur le poste client.

```
ssh-keygen -t rsa -b 4096 -C "[your github's email]"
```

Ensuite lorsque le terminal affiche ce texte appuyez sur Entrer

```
> Entrez un fichier dans lequel enregistrer la clé
(/Users/you/.ssh/id_rsa) : [Appuyez sur Entrer]
```

Finalement il faut définir un mot de passe associé à cette clé

```
> Entrez la phrase secrète (vide pour ne pas utiliser de phrase secrète) : [Tapez une phrase secrète]
> Entrez à nouveau la même phrase secrète : [Tapez à nouveau la phrase secrète]
```

Par défaut la clé publique et privée sont enregistrée dans le répertoire *.ssh*