

**in104**

PROJET D'INFORMATIQUE

---

Florence Carton

April 10. 2018

ENSTA ParisTech

## 1. GIT

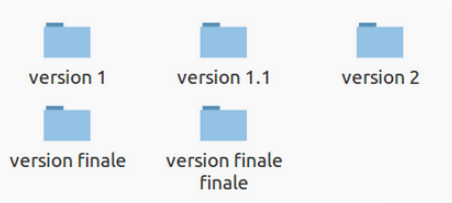
Les bases

Pour les plus avancés

## 2. Python

**git**





**Figure 1:** slides from Antonin Raffin <sup>1</sup>

---

<sup>1</sup><http://slides.com/antoninraffin/git>

## Git = gestionnaire de version

- Qui a fait quelle modif et pourquoi ?
- Sauvegarde chaque modif
- Permet à chacun de travailler sur le meme fichier
- Chaque développeur a une copie en local

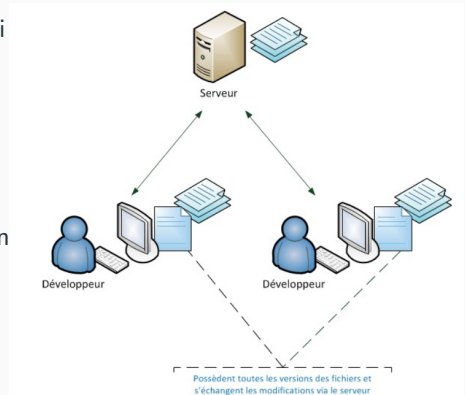


Figure 2: GIT <sup>a</sup>

<sup>a</sup> [www.openclassrooms.com/courses/gerer-son-code-avec-git-et-github](http://www.openclassrooms.com/courses/gerer-son-code-avec-git-et-github)

- clef ssh

1. générer une clef (mettre tout par défaut)

```
$ ssh-keygen -t rsa -C "florence.carton@ensta-paristech.fr"
```

2. afficher la clef ssh générée

```
$ cat ~/.ssh/id_rsa.pub
```

3. coller la clef générée dans la section 'My SSH Keys'

Une clef est nécessaire par ordinateur

- config

```
$ git config --global user.name "Carton Florence"
```

```
$ git config --global user.email "florence.carton@ensta-paristech.fr"
```

# créer un projet

1. Créer un dossier dans votre ordinateur et l'initialiser

```
$ mkdir dossier_du_projet  
$ cd dossier_du_projet  
$ git init
```

2. créer un nouveau projet dans git
3. ajouter un premier fichier

```
$ touch README.md  
$ git add README.md  
$ git commit -m "first commit"
```

4. lier votre dossier local au projet sur git

```
$ git remote add origin git@gitlab.ensta.fr:fcarton/projet.git
```

5. "pousser" le readme sur le git

```
$ git push -u origin master
```

A ce niveau la un projet est créé et initialisé.

Tout personne rejoignant ce projet devra être ajoutée à la liste des membres, et aura simplement à effectuer :

```
$ git clone lien_ssh_du_projet  
$ git clone git@gitlab.ensta.fr:fcarton/projet.git
```

Et le dossier sera créé sur son ordinateur (à l'endroit où il se trouve quand il a lancé la ligne de commande)



- Add: ajouter un ou des fichier(s) pour le prochain commit

```
$ git add mon_fichier1 mon_fichier2  
$ git add --all
```

- Commit : Commit les fichiers ajoutés précédemment

```
$ git commit -m 'Commentaire sur les changements effectués '
```

- Pull : récupérer ce que les autres membres du projet ont fait

```
$ git pull
```

- Push : pousser tous les commits sur le git

```
$ git push
```

Exemple : 2 bugs à résoudre :

bug 1 : fichier a.py et b.py modifiés → bug 1 résolu

bug 2 : fichier c.py → bug 2 résolu

```
$ git add a.py b.py
$ git commit -m 'bug 1 solved !'
$ git add c.py
$ git commit -m 'bug 2 solved !'
$ git pull
$ git push
```

- Status: affiche le status du dossier local git (fichiers modifiés, à ajouter...)

```
$ git status
```

Note : toujours faire un pull avant un push !!

# les branches

- Lister les branches

```
$ git branch
* master
```

- Créer une branche

```
$ git branch my_new_branch
$ git branch
* master
  my_new_branch
```

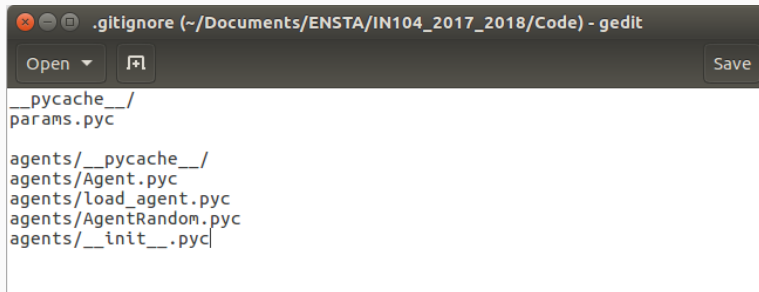
- Se déplacer dans my\_new\_branch

```
$ git checkout my_new_branch
```

- Fusionner les branches

```
$ git checkout master
$ git merge my_new_branch
```

# ignorer les fichiers




The image shows a screenshot of a gedit text editor window. The title bar at the top reads ".gitignore (~/Documents/ENSTA/IN104\_2017\_2018/Code) - gedit". Below the title bar, there is a toolbar with an "Open" button (with a dropdown arrow) and a "Save" button. The main text area contains the following lines of text, which are entries for a .gitignore file:


```
__pycache__/  
params.pyc  
  
agents/__pycache__/  
agents/Agent.pyc  
agents/load_agent.pyc  
agents/AgentRandom.pyc  
agents/__init__.pyc
```




GIT BLAME

 Catkin\_OpenCV.txt 2.18 KB

[Edit](#) [Raw](#) [Blame](#) [History](#) [Permalink](#) [Remove](#)

 Catkin\_OpenCV.txt 2.18 KB

[Edit](#) [Raw](#) [Normal View](#) [History](#) [Permalink](#) [Remove](#)

e85d9861  Carton Florence first tests with ...

1 Catkin avec OpenCV

2

3


4 Creation du catkin workspace

5


6 \$ mkdir -p ~/catkin\_ws/src  
7 (ou \$ mkdir /catkin\_ws/src)

8 \$ cd /catkin\_ws/src  
9 \$ catkin\_init\_workspace

10

20f78866  Carton Florence 1er tests avec op...

11 Creation d'un package catkin (tjs dans le dossier src de catkin\_ws)

e85d9861  Carton Florence first tests with ...

12

13 \$ catkin\_create\_pkg ardrone\_vision std\_msgs roscpp

14

15 ardrone\_vision : name of the package

# merge conflict

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

fichier test\_file.md

```
<<<<<< HEAD
# some modifications here to make conflicts
=====
# blablabla !
>>>>>> my_branch
```

HEAD : modif de la branche master

my\_branch : modif de my\_branch (modif des autres)

Une fois les problèmes résolus :

```
$ git add test_file.md
$ git commit -m'conflic in test_file.md solved'
$ git push
```



- Abandonner les changements d'un fichier

```
$ git checkout — my_file
```

- Annuler les changements du dernier commit

```
$ git revert
```

- Oh shit git ! <http://ohshitgit.com/>
- How to undo (almost) everything in git <https://blog.github.com/2015-06-08-how-to-undo-almost-anything-with-git/>
- Openclassroom : Gérer vos codes sources avec Git et Github :  
[www.openclassrooms.com/courses/gerer-son-code-avec-git-et-github](http://www.openclassrooms.com/courses/gerer-son-code-avec-git-et-github)
- Slides Antonin Raffin : <http://slides.com/antoninraffin/>

Questions ?

**In case of fire**



1. git commit



2. git push



3. leave building

**pyhton**



# les classes en python

*fichier animal.py*

```
class Animal():  
    def __init__(self, name, type):  
        self.type = type  
        self.name = name  
    def get_name(self):  
        return self.name
```

# les classes en python

*fichier animal.py*

```
class Animal():  
    def __init__(self, name, type):  
        self.type = type  
        self.name = name  
    def get_name(self):  
        return self.name
```

*fichier main.py*

```
from animal import Animal  
mon_chat = Animal('felix', 'cat')  
nom_de_mon_chat = mon_chat.get_name()  
print('mon chat s'appelle ', nom_de_mon_chat)
```

```
$ python3 main.py  
$ mon chat s'appelle felix
```

# les classes en python

*fichier animal.py*

```
class Animal():  
    def __init__(self, name, type):  
        self.type = type  
        self.name = name  
    def get_name(self):  
        return self.name
```

*fichier main.py*

```
from animal import Animal  
mon_chat = Animal('felix', 'cat')  
nom_de_mon_chat = mon_chat.get_name()  
print('mon chat s'appelle ', nom_de_mon_chat)
```

*fichier cat.py*

```
from animal import Animal  
class Cat(Animal):  
    def __init__(self, name, color):  
        Animal.__init__(self, name, 'cat')  
        self.color = color  
    def miaow(self):  
        print('Miaou !')
```

```
$ python3 main.py  
$ mon chat s'appelle felix
```

# les classes en python

*fichier animal.py*

```
class Animal():  
    def __init__(self, name, type):  
        self.type = type  
        self.name = name  
    def get_name(self):  
        return self.name
```

*fichier main.py*

```
from animal import Animal  
mon_chat = Animal('felix', 'cat')  
nom_de_mon_chat = mon_chat.get_name()  
print('mon chat s'appelle ', nom_de_mon_chat)
```

```
$ python3 main.py  
$ mon chat s'appelle felix
```

*fichier cat.py*

```
from animal import Animal  
class Cat(Animal):  
    def __init__(self, name, color):  
        Animal.__init__(self, name, 'cat')  
        self.color = color  
    def miaow(self):  
        print('Miaou !')
```

*fichier main.py*

```
from cat import Cat  
mon_chat = Cat('felix', 'black')  
nom_de_mon_chat = mon_chat.get_name()  
print('mon chat s'appelle ', nom_de_mon_chat)  
mon_chat.miaow()
```

```
$ python3 main.py  
$ mon chat s'appelle felix  
$ Miaou !
```



Questions?