

Sommaire

git	2
1. Installer git	2
1.1. Installation	2
1.2. Configurer git	3
2. Gérer des branches en local.....	4
2.1 Créer un projet git et effectuer : "commit initial"	4
2.2 Travailler avec git : "premières modifications".....	6
3. Créer une nouvelle branche du projet.....	8
3.1. Créer une branche et travailler dessus : création d'une nouvelle branche.....	8
3.2. Travailler aussi sur la branche master	10
4. Réconcilier une branche avec master	12
4.1. Fusionner la branche evo sur la branche master : la commande Merge	12
4.2. Fusionner la branche evo sur la branche master : la commande Rebase	13
5. Configurer un dépôt distant sur github	15
5.1. Créez votre compte sur github :.....	15
5.2. Créez un nouveau dépôt github.....	15
5. Commandes spécifiques à un dépôt distant.....	17
6. Documenter un développement, paramétrage supplément.....	18
6.1 Le fichier readme.md	18
6.2 Le langage MarkDown.....	19
6.3. Ignorer le suivi de certains fichiers : .le fichier .gitignore	19
7. Travailler à plusieurs en réseau sur un dépôt commun.....	20
scénario.....	20
a) Création d'un dépôt commun.....	20
b) Chaque développeur travaille sur sa partie–	20
c) Il est temps faire profiter les autres de son travail.....	20
d) Tout le monde doit récupérer le travail des autres.....	20
e) Tous les développeurs travaille sur une même partie.....	20
8. les autres commandes utiles.....	21

Objectifs :

- ✓ Installer git,
- ✓ Travailler avec git sur la gestion de branches en local,
- ✓ Configurer git pour un dépôt distant,
- ✓ Documenter un dépôt git,
- ✓ Travailler avec git avec un dépôt distant (remote) à plusieurs.

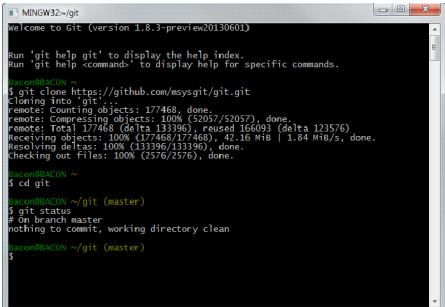
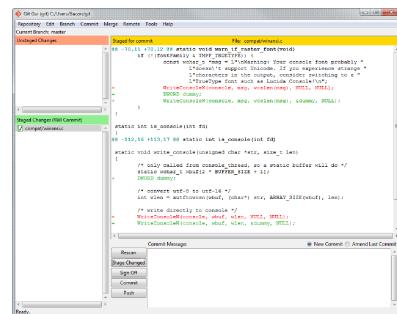
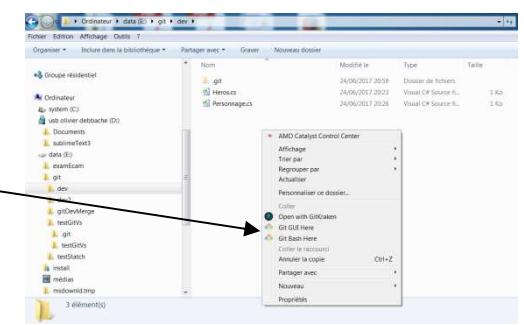
1. Installer git

1.1. Installation

ou	
https://git-for-windows.github.io/	https://git-scm.com/downloads

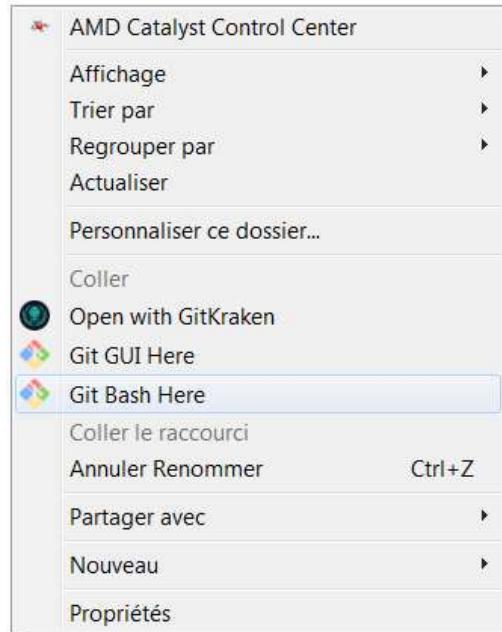
git est lui-même géré dans un dépôt git hub, Installez le. Il fournit trois fonctionnalités principales :

- git sous la forme d'un interpréteur de commandes shell que nous allons utiliser intensivement,
- git avec les commandes de base intégrées dans un environnement graphique,
- git intégré dans l'explorateur de fichier windows.

git Bash	
git Gui	
intégration shell	

1.2. Configurer git

Créez un répertoire dev vide,
placez-vous dedans
et lancez le menu contextuel :
Git Bash Here



On va configurer l'environnement
git config --global user.name "prenom et nom"
git config --global user.mail
prenom.nom@gmail.com
git config --global core.autocrlf true
git config --global core.safecrlf true

```
MINGW64:/e/git/dev
odeb@odeb-ws MINGW64 /e/git/dev
$ git config --global user.name odeb
odeb@odeb-ws MINGW64 /e/git/dev
$ git config --global user.mail olivier.debbache@gmail.com
odeb@odeb-ws MINGW64 /e/git/dev
$ git config --global core.autocrlf true
odeb@odeb-ws MINGW64 /e/git/dev
$ git config --global core.safecrlf true
```

Vous connaissez maintenant les commandes :

- ✓ git config --global qui permet de configurer les options de l'environnement pour tous vos dépôts locaux.
- ✓ git config --local configure les options du dépôt local en cours
- ✓ git config --list --local permet de consulter les paramètres du dépôt local et
- ✓ git config --list --global permet de consulter les paramètres généraux de git

Tout paramètre local existe au niveau global.

2. Gérer des branches en local

2.1 Créer un projet git et effectuer : "commit initial"

Dans le répertoire dev.

Dans ce répertoire, créer deux classes cSharp :

Héros.cs et

Personnage.cs

On utilisera un éditeur quelconque plutôt que VS.

Vous pouvez aussi utiliser vi qui est directement intégré au Shell Git.

Initialiser le dépôt git : git init.

Contrôler l'état : git status.

git init crée un répertoire .git dans le répertoire où la commande a été lancée. A partir de ce moment toute modification dans le répertoire et son arborescence est tracée dans git.

L'historique des manipulations et le paramétrage de votre dépôt git est conservé dans ce répertoire .git.

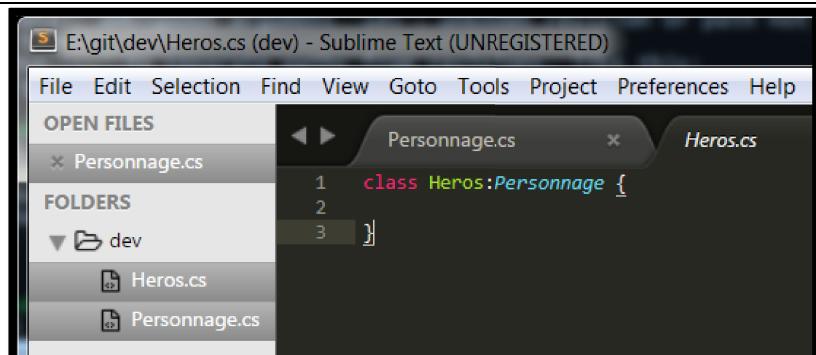
vous pouvez le visualiser tous ses fichiers, tout est au format texte.

Ajoutez les fichiers au contrôle de validation :

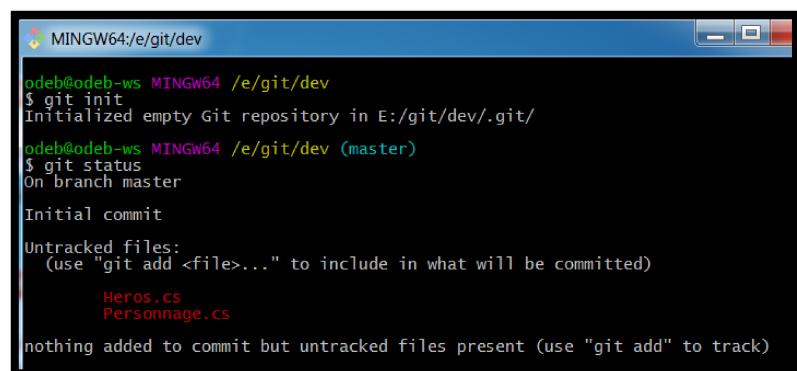
git add *

Contrôler l'état :

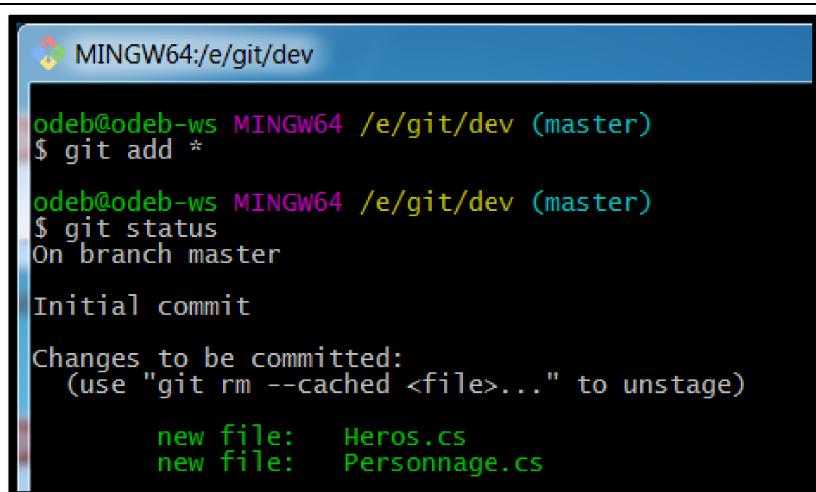
git status.



```
E:\git\dev\Heros.cs (dev) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
OPEN FILES Personnage.cs * Heros.cs
FOLDERS dev
  Heros.cs
  Personnage.cs
```



```
MINGW64:/e/git/dev
odeb@odeb-ws MINGW64 /e/git/dev
$ git init
Initialized empty Git repository in E:/git/dev/.git/
odeb@odeb-ws MINGW64 /e/git/dev (master)
$ git status
On branch master
Initial commit
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Heros.cs
    Personnage.cs
nothing added to commit but untracked files present (use "git add" to track)
```



```
MINGW64:/e/git/dev
odeb@odeb-ws MINGW64 /e/git/dev (master)
$ git add *
odeb@odeb-ws MINGW64 /e/git/dev (master)
$ git status
On branch master
Initial commit
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   Heros.cs
    new file:   Personnage.cs
```

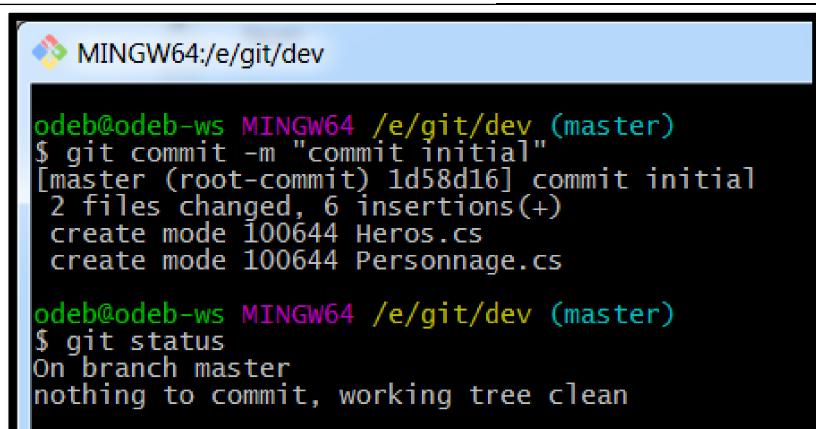
Committer les modifications :

git commit -m "commit initial"

Contrôler l'état : git status.

Le message du commit est important et décrit la modification effectuée sur le source.

Il vaut mieux multiplier les commit : un commit = un lot de modifications cohérentes sur un source.



```
MINGW64:/e/git/dev
odeb@odeb-ws MINGW64 /e/git/dev (master)
$ git commit -m "commit initial"
[master (root-commit) 1d58d16] commit initial
  2 files changed, 6 insertions(+)
  create mode 100644 Heros.cs
  create mode 100644 Personnage.cs
odeb@odeb-ws MINGW64 /e/git/dev (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Visualiser l'historique des modifications :
git log

Chaque commit possède un identifiant unique : ici 1d58...

On en a besoin pour manipuler les commit, heureusement chaque fois que l'on doit saisir ce code, git arrive à reconnaître le commit tant que l'id saisi tant que celui-ci est discriminant par rapport à tous les autres commit existants.

Ainsi j'ai le droit de taper la commande
git log 1d58 il comprend que je veux manipuler le commit
1d58d16f811a55fa224878f4109ee453a951
1133 ouf.

```
MINGW64:/e/git/dev
odeb@odeb-ws MINGW64 /e/git/dev (master)
$ git log
commit 1d58d16f811a55fa224878f4109ee453a9511133
Author: odeb <olivier.debbache@gmail.com>
Date:   Sun Jun 25 10:39:47 2017 +0200

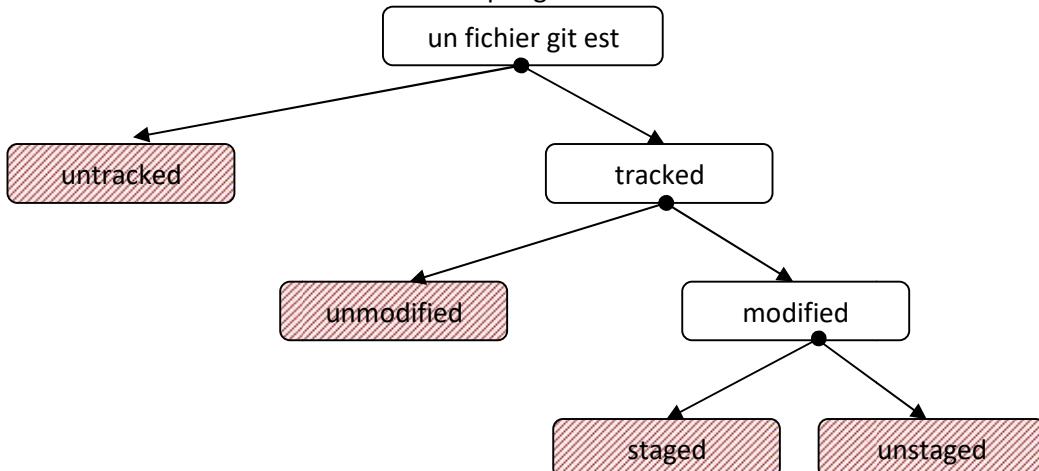
    commit initial
```

Les principes qu'il faut retenir pour comprendre comment git gère les sources d'un développement :

git gère trois lieux dans un dépôt, chaque lieu correspond à un état du source modifié :	working directory	staging area ou index	local repository ou dépôt ou repo
les fichiers sont à l'état :	unstaged	staged	unmodified

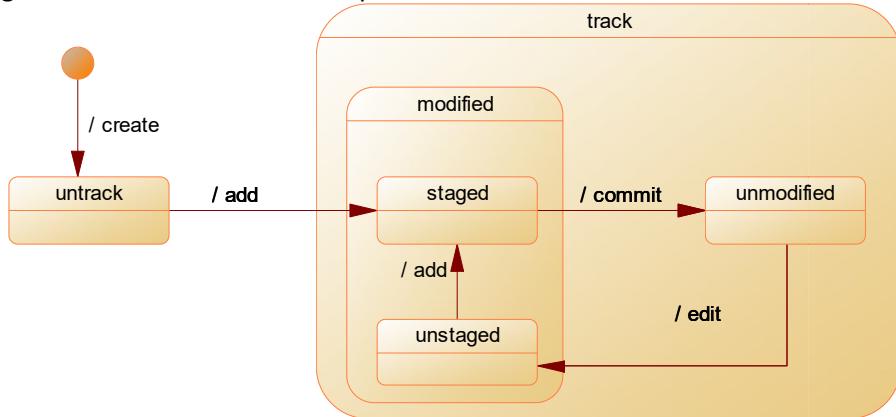
les commandes add et commit permettent de gérer l'état de chacun des fichiers. à noter que le premier appel à la commande add est un peu spécifique puisqu'elle permet de placer le fichier dans l'état tracked : suivi par git. la modification d'un source le place un fichier à l'état unstaged	working directory	staging area ou index	local repository ou dépôt ou repo
les fichiers sont à l'état :	unstaged	staged	unmodified

On peut résumer l'ensemble des états d'un fichier suivi par git de la manière suivante :



En rouge les états concrets les autres sont des sur-états qu'on ne voit pas apparaître directement.

Le diagramme état transition correspondant :



2.2 Travailler avec git : "premières modifications"

Ajoutez une méthode à la classe Personnage.

```

E:\git\dev\Personnage.cs • (dev) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
OPEN FILES
Personnage.cs
FOLDERS
dev
Heros.cs
Personnage.cs

```

```

class Personnage {
    void frapper(Personnage personnage, int force){
}
}

```

A screenshot of Sublime Text showing the file 'Personnage.cs' with the following code:

```

class Personnage {
    void frapper(Personnage personnage, int force){
}
}

```

Contrôler l'état : git status

Propagez les modifications :

git add, git commit

```

MINGW64:/e/git/dev
odeb@odeb-ws MINGW64 /e/git/dev (master)
$ git status
On branch master
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:   Personnage.cs

no changes added to commit (use "git add" and/or "git commit -a")

odeb@odeb-ws MINGW64 /e/git/dev (master)
$ git add *

odeb@odeb-ws MINGW64 /e/git/dev (master)
$ git commit -m "ajout méthode frapper à personnage"
[master ce62e01] ajout méthode frapper à personnage
 1 file changed, 3 insertions(+), 1 deletion(-)

```

A screenshot of a terminal window showing the output of 'git status' and 'git commit' commands:

```

MINGW64:/e/git/dev
odeb@odeb-ws MINGW64 /e/git/dev (master)
$ git status
On branch master
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:   Personnage.cs

no changes added to commit (use "git add" and/or "git commit -a")

odeb@odeb-ws MINGW64 /e/git/dev (master)
$ git add *

odeb@odeb-ws MINGW64 /e/git/dev (master)
$ git commit -m "ajout méthode frapper à personnage"
[master ce62e01] ajout méthode frapper à personnage
 1 file changed, 3 insertions(+), 1 deletion(-)

```

Créez une nouvelle classe : Monstre.cs

```

E:\git\dev\Monstre.cs (dev) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
OPEN FILES
Personnage.cs
Monstre.cs
FOLDERS
dev
Heros.cs
Monstre.cs
Personnage.cs

```

```

class Monstre:Personnage {
}

```

A screenshot of Sublime Text showing the file 'Monstre.cs' with the following code:

```

class Monstre:Personnage {
}

```

Contrôler l'état : git status

Propagez les modifications :

git add,

git commit

```
MINGW64:/e/git/dev
odeb@odeb-ws MINGW64 /e/git/dev (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    Monstre.cs

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

odeb@odeb-ws MINGW64 /e/git/dev (master)
$ git add *

odeb@odeb-ws MINGW64 /e/git/dev (master)
$ git commit -m "Création de la class Monstre"
[master 511988a] Création de la class Monstre
 1 file changed, 3 insertions(+)
 create mode 100644 Monstre.cs
```

Complétez le graphe des commits avec le n° de chaque commit



commit

commit initial

commit

ajout méthode frapper à Personnage

commit

création de la class Monstre

Vous devez maîtriser et comprendre l'effet des commandes :

- ✓ git init
- ✓ git add
- ✓ git commit
- ✓ git status
- ✓ git log
- ✓ les différents états des documents gérés par git
- ✓ les différentes transitions entre les états

3. Créer une nouvelle branche du projet

Une fonctionnalité de git permet de créer des branches de développement qui permettent de séparer un développement temporairement du développement principal : test spécifique, évolution, réorganisation. Ces branches partent bien sur du développement existant.

3.1. Créer une branche et travailler dessus : création d'une nouvelle branche

Nous voulons faire évoluer l'application sans impacter le développement actuel : pour cela nous allons créer une branche evo.

Créer la branche evo :

```
git branch evo
```

la branche est créée à partir de l'endroit où nous trouvons.

C'est une bifurcation de la branche master. evo contient déjà tout ce qui a été fait sur master.

Donner la liste des branches du projet :

```
git branch
```

une étoile est placée devant la branche où l'on se situe, ici master.

```
MINGW64:/e/git/dev
odeb@R108P01 MINGW64 /e/git/dev (master)
$ git branch evo
```

Pour se déplacer de branche en branche :

```
git checkout evo
```

```
MINGW64:/e/git/dev
odeb@R108P01 MINGW64 /e/git/dev (master)
$ git checkout evo
Switched to branch 'evo'

odeb@R108P01 MINGW64 /e/git/dev (evo)
$ git branch
* evo
  master
```

Positionnez vous sur la branche evo si vous n'êtes pas dessus

modifier la classe Personnage : ajouter la méthode Regarder à la classe Personnage.

consultez l'historique

```
E:\git\dev\Personnage.cs (dev) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
OPEN FILES
  Personnage.cs
  Arme.cs
FOLDERS
  dev
    Arme.cs
    Heros.cs
    Monstre.cs
    Personnage.cs
Personnage.cs
1 class Personnage {
2     void frapper(Personnage personnage, int force){
3
4
5         public Lieu Regarder(Direction direction, int distance) {
6
7
8         }
9     }
}
```

validez l'évolution (sur cette branche) :
git add, git commit

```
MINGW64:/e/git/dev
$ git add Personnage.cs
odeb@odeb-ws MINGW64 /e/git/dev (evo)
$ git status
On branch evo
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

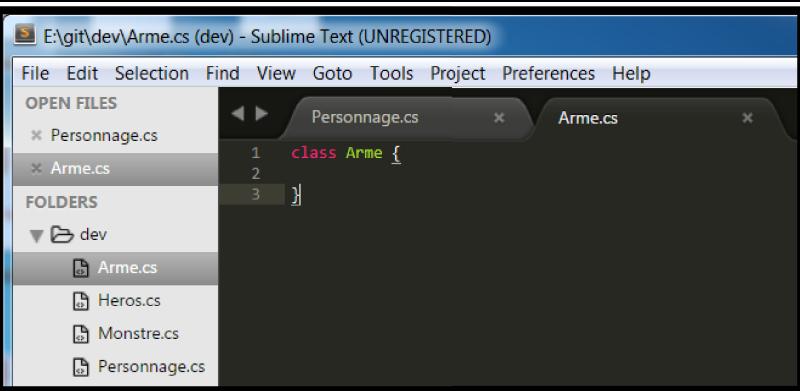
    modified:   Personnage.cs

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

    Arme.cs

odeb@odeb-ws MINGW64 /e/git/dev (evo)
$ git commit -m "Ajout méthode Regarder à la classe Personnage"
[evo 60d0532] Ajout méthode Regarder à la classe Personnage
 1 file changed, 4 insertions(+)
```

ajouter une nouvelle classe Arme.cs



validez l'évolutions (sur cette branche) :
git add, git commit

```
MINGW64:/e/git/dev
odeb@odeb-ws MINGW64 /e/git/dev (evo)
$ git add Arme.cs

odeb@odeb-ws MINGW64 /e/git/dev (evo)
$ git status
On branch evo
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   Arme.cs

odeb@odeb-ws MINGW64 /e/git/dev (evo)
$ git commit -m "Création de la class Arme"
[evo a1fda96] Création de la class Arme
 1 file changed, 3 insertions(+)
 create mode 100644 Arme.cs
```

Repositionnez-vous sur master :
git checkout master
visualiser les sources, vous constaterez que les deux commits n'existent pas dans master :

- Arme.cs est absent,
- Personnage.cs ne contient pas de méthode Regarder

```
MINGW64:/e/git/dev
odeb@odeb-ws MINGW64 /e/git/dev (evo)
$ git checkout master
Switched to branch 'master'

odeb@odeb-ws MINGW64 /e/git/dev (master)
$ ls
Heros.cs  Monstre.cs  Personnage.cs

odeb@odeb-ws MINGW64 /e/git/dev (master)
$ cat personnage.cs
class Personnage {
    void frapper(Personnage personnage, int force){
    }
```

3.2. Travailler aussi sur la branche master

On se repositionne sur la branche principale master si ce n'est pas déjà fait.

Ajoutez la méthode Deplacer à personnage J'en profite pour mettre un F majuscule au nom de la méthode frapper et le mot clé public que j'avais oublié.

E:\git\dev\Personnage.cs (dev) - Sublime Text (UNREGISTERED)

```
File Edit Selection Find View Tools Project Preferences Help
OPEN FILES
  Personnage.cs
Arme.cs
FOLDERS
  dev
    Heros.cs
    Monstre.cs
    Personnage.cs
Personnage.cs
```

```
1 class Personnage {
2     public void Frapper(Personnage personnage, int force){
3         }
4     }
5     public void Deplacer(Direction direction) {
6         }
7     }
8 }
```

Après avoir enregistré mon source :
je lance la commande
git diff Personnage.cs
j'obtiens alors les différences de source entre
ce qui se trouve dans la working dir et mon
dépôt validé.

```
MINGW64:/e/git/dev
odeb@odeb-ws MINGW64 /e/git/dev (master)
$ git diff Personnage.cs
diff --git a/Personnage.cs b/Personnage.cs
index 9349cea..3c47e46 100644
--- a/Personnage.cs
+++ b/Personnage.cs
@@ -1,5 +1,9 @@
 class Personnage {
-    void frapper(Personnage personnage, int force){
+    public void Frapper(Personnage personnage, int force){

        }

+    public void Deplacer(Direction direction) {
+        }
}
\ No newline at end of file
```

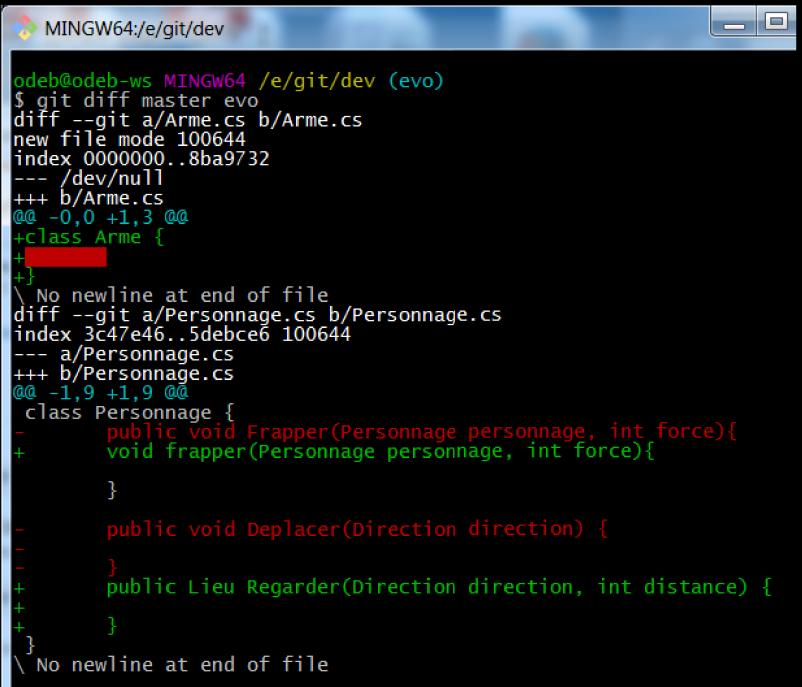
validez les évolutions
Consultez l'historique

```
MINGW64:/e/git/dev
$ git status
On branch master
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:   Personnage.cs

no changes added to commit (use "git add" and/or "git commit -a")
odeb@odeb-ws MINGW64 /e/git/dev (master)
$ git add *
odeb@odeb-ws MINGW64 /e/git/dev (master)
$ git commit -m "class Personnage ajout méthode Deplacer et reécriture
de la signature de Frapper"
[master e59e32e] class Personnage ajout méthode Deplacer et reécriture
de la signature de Frapper
 1 file changed, 5 insertions(+), 1 deletion(-)
```

Positionnez vous sur la branche evo et récapitulez les différences entre master et evo.



```
odeb@odeb-ws MINGW64 /e/git/dev (evo)
$ git diff master evo
diff --git a/Arme.cs b/Arme.cs
new file mode 100644
index 000000..8ba9732
--- /dev/null
+++ b/Arme.cs
@@ -0,0 +1,3 @@
+class Arme {
+}
+
\ No newline at end of file
diff --git a/Personnage.cs b/Personnage.cs
index 3c47e46..5debce6 100644
--- a/Personnage.cs
+++ b/Personnage.cs
@@ -1,9 +1,9 @@
 class Personnage {
-    public void Frapper(Personnage personnage, int force){
+    void frapper(Personnage personnage, int force){
        }
-
-    public void Deplacer(Direction direction) {
-
+    public Lieu Regarder(Direction direction, int distance) {
-
    }
\ No newline at end of file
```

Vous devez maîtriser et comprendre l'effet des commandes :

- ✓ git branch
- ✓ git checkout
- ✓ git diff

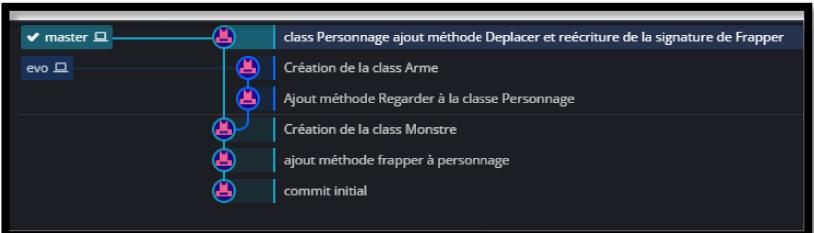
Attention : Dupliquez le répertoire dev en devMerge et devRebase : nous allons étudier les conséquences de deux commandes équivalentes qui n'ont pas le même effet sur la gestion du source.

Vous devez donc avoir 3 répertoires identiques.

4. Réconcilier une branche avec master

4.1. Fusionner la branche evo sur la branche master : la commande Merge

Rappel l'arborescence git à cet état ci :



On va travailler dans le répertoire devMerge. Redémarrer un interpréteur de commande sur ce répertoire.

Maintenant, on suppose que ce qui a été développé sur chacune des deux branches master et evo est à conserver :

On va donc fusionner ces deux branches : la branche evo sur la branche master.

Se positionner sur master (sinon on risque de faire l'opération à l'envers) par checkout, Fusionner : git merge evo

Problème personnage.cs a été modifié dans les deux branches : master et evo. Il faut résoudre le conflit. git ne peut pas faire de choix à votre place.

On va s'assurer que les modifications réalisées sur les 2 branches ont été conservées

Voici le code en conflit :

```
<<<<< HEAD
// partie du code provenant de
master(HEAD)
=====
// partie du code provenant de la branche
evo
>>>>>
```

Je crois qu'il faut mieux supprimer les tags <<<< ===== >>>>> sinon git croit qu'il existe toujours un conflit...

Au résultat on obtient :

```
MINGW64:/e/git/devMerge
odeb@odeb-ws MINGW64 /e/git/devMerge (evo)
$ git checkout master
Switched to branch 'master'
odeb@odeb-ws MINGW64 /e/git/devMerge (master)
$ git merge evo
Auto-merging Personnage.cs
CONFLICT (content): Merge conflict in Personnage.cs
Automatic merge failed; fix conflicts and then commit the result.
odeb@odeb-ws MINGW64 /e/git/devMerge (master|MERGING)
$ vi personnage.cs
```

```
MINGW64:/e/git/devMerge
class Personnage {
    public void Frapper(Personnage personnage, int force){
    }
<<<<< HEAD
    public void Deplacer(Direction direction) {
    }
===== public Lieu Regarder(Direction direction, int distance) {
    }
>>>>> evo
```

```
MINGW64:/e/git/devMerge
class Personnage {
    public void Frapper(Personnage personnage, int force){
    }
    public void Deplacer(Direction direction) {
    }
    public Lieu Regarder(Direction direction, int distance) {
    }
}
```

On sauvegarde et on a plus qu'à : git add git

commit...

Conflit résolu...

```
MINGW64:/e/git/devMerge
odeb@odeb-ws MINGW64 /e/git/devMerge (master|MERGING)
$ git status
On branch master
All conflicts fixed but you are still merging.
(use "git commit" to conclude merge)

Changes to be committed:
  new file:  Arme.cs

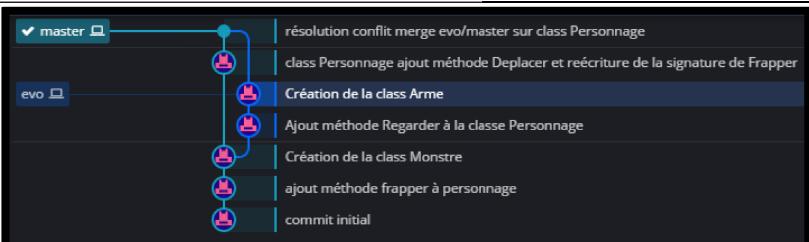
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:  Personnage.cs

odeb@odeb-ws MINGW64 /e/git/devMerge (master|MERGING)
$ git add Personnage.cs

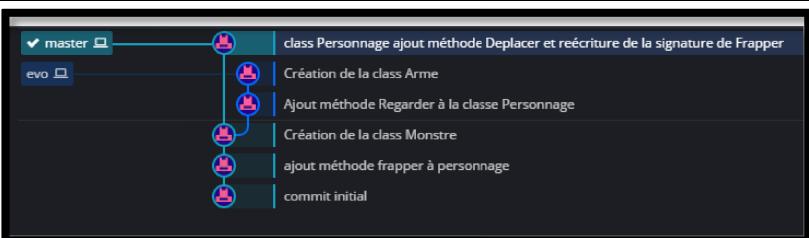
odeb@odeb-ws MINGW64 /e/git/devMerge (master|MERGING)
$ git commit -m "résolution conflit merge evo/master sur class Personnage"
[master c131825] résolution conflit merge evo/master sur class Personnage
```

Notre arborescence ressemble maintenant à ceci :



4.2. Fusionner la branche evo sur la branche master : la commande Rebase

Rappel l'arborescence git à cet état ci :



On va travailler dans le répertoire devRebase.

On se positionne sur evo

git rebase master

rebase récupère toute l'arborescence jusqu'où il y a eu une disjonction et la repositionne après master

```
MINGW64:/e/git/devRebase
odeb@odeb-ws MINGW64 /e/git/devRebase (evo)
$ git checkout evo
Already on 'evo'

odeb@odeb-ws MINGW64 /e/git/devRebase (evo)
$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: Ajout méthode Regarder à la classe Personnage
error: Failed to merge in the changes.
Using index info to reconstruct a base tree...
M   Personnage.cs
 Falling back to patching base and 3-way merge...
Auto-merging Personnage.cs
CONFLICT (content): Merge conflict in Personnage.cs
Patch failed at 0001 Ajout méthode Regarder à la classe Personnage
The copy of the patch that failed is found in: .git/rebase-apply/patch

When you have resolved this problem, run "git rebase --continue".
If you prefer to skip this patch, run "git rebase --skip" instead.
To check out the original branch and stop rebasing, run "git rebase --abort".

odeb@odeb-ws MINGW64 /e/git/devRebase (evo|REBASE 1/2)
$ |
```

à nouveau conflit.

```
MINGW64:/e/git/devRebase
class Personnage {
    public void Frapper(Personnage personnage, int force){
    }
}

<<<<< HEAD
    public void Deplacer(Direction direction) {
    }
=>>>
    public Lieu Regarder(Direction direction, int distance) {
    }
}
>>>>> Ajout méthode Regarder à la classe Personnage
```

même solution que précédemment

```
MINGW64:/e/git/devRebase
class Personnage {
    public void Frapper(Personnage personnage, int force){
    }
    public void Deplacer(Direction direction) {
    }
    public Lieu Regarder(Direction direction, int distance) {
    }
}
```

On sauvegarde et on a plus qu'à :

git add

Attention pas de commit mais git rebase --continue

git branch -f master

git checkout master

conflit résolu.

Doit y avoir plus simple mais je n'ai pas trouvé.

```
MINGW64:/e/git/devRebase
odeb@odeb-ws MINGW64 /e/git/devRebase (evo|REBASE 1/2)
$ git add Personnage.cs

odeb@odeb-ws MINGW64 /e/git/devRebase (evo|REBASE 1/2)
$ git status
rebase in progress; onto e59e32e
You are currently rebasing branch 'evo' on 'e59e32e'.
  (all conflicts fixed: run "git rebase --continue")

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   Personnage.cs

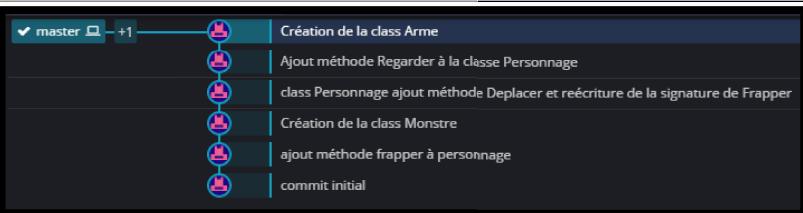
odeb@odeb-ws MINGW64 /e/git/devRebase (evo|REBASE 1/2)
$ git rebase --continue
Applying: Ajout méthode Regarder à la classe Personnage
Applying: Création de la class Arme

odeb@odeb-ws MINGW64 /e/git/devRebase (evo)
$ git branch -f master

odeb@odeb-ws MINGW64 /e/git/devRebase (evo)
$ git checkout master
Already on 'master'

odeb@odeb-ws MINGW64 /e/git/devRebase (master)
```

Notre arborescence ressemble maintenant à cela : toute linéaire



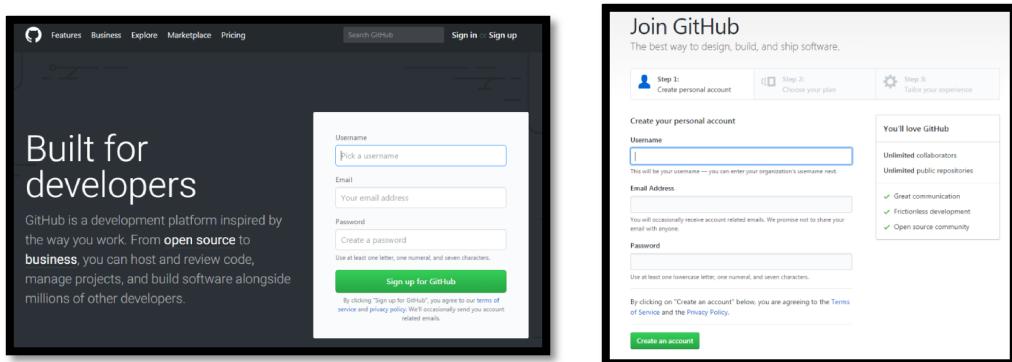
Vous connaissez maintenant :

- ✓ git merge
- ✓ git rebase
- ✓ git rebase --continue
- ✓ git branch -f

5. Configurer un dépôt distant sur github

5.1. Créez votre compte sur github :

- ✓ Utilisez une adresse mail cohérente : prenom.nom@gmail.com : pas de iloveRockNroll95@latuvu.ty merci.
- ✓ Le but est de créer votre portfolio pas de vous inscrire à un club de rencontre.



5.2. Créez un nouveau dépôt github

Créez un nouveau dépôt vide sous github

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: odebba / Repository name: sioCrawler ✓

Great repository names are short and memorable. Need inspiration? How about [bookish-guide](#).

Description (optional): sio crawler

Public: Anyone can see this repository. You choose who can commit.

Private: You choose who can see and commit to this repository.

Initialize this repository with a README: This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: None | ⓘ

Create repository

on va utiliser la possibilité de "pusher" un dépôt existant

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH <https://github.com/odebb/sioCrawler.git>

We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# sioCrawler" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/odebb/sioCrawler.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/odebb/sioCrawler.git
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

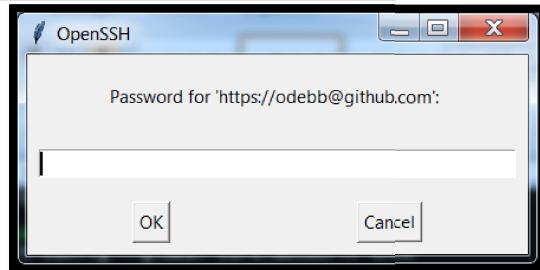
si vous ne l'avez pas déjà fait : configuez votre environnement local

```
MINGW64:/e/git/devMerge
odeb@odeb-ws MINGW64 /e/git/devMerge (master)
$ git config user.name "odebb"
odeb@odeb-ws MINGW64 /e/git/devMerge (master)
$ git config user.email olivier.debbache@gmail.com
odeb@odeb-ws MINGW64 /e/git/devMerge (master)
$ git config --global core.autocrlf true
odeb@odeb-ws MINGW64 /e/git/devMerge (master)
$ git config --global core.safecrlf true
odeb@odeb-ws MINGW64 /e/git/devMerge (master)
$ git remote add origin https://github.com/odebb/sioCrawler.git
odeb@odeb-ws MINGW64 /e/git/devMerge (master)
$ git remote -v
origin https://github.com/odebb/sioCrawler.git (fetch)
origin https://github.com/odebb/sioCrawler.git (push)
```

Il ne reste plus qu'à pousser votre dépôt local vers le dépôt distant hébergé chez git hub.

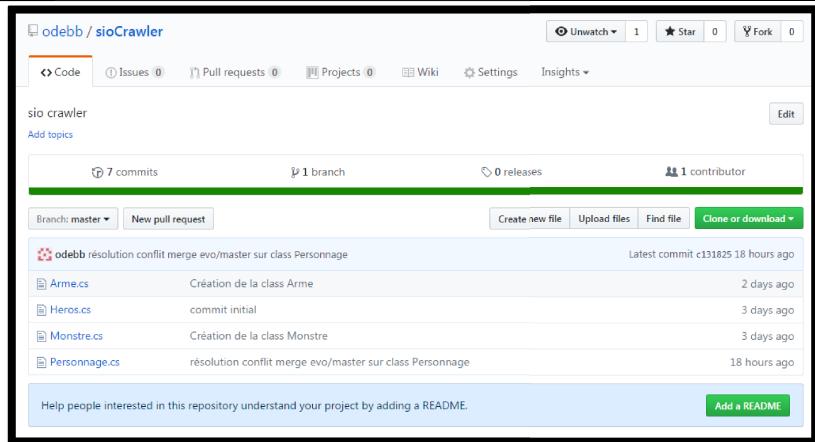
```
MINGW64:/e/git/devMerge
odeb@odeb-ws MINGW64 /e/git/devMerge (master)
$ git push -u origin master
Username for 'https://github.com': odeb
```

saisissez votre mot de passe git hub



Voilà votre code est au chaud.

```
MINGW64:/e/git/devMerge
odeb@odeb-ws MINGW64 /e/git/devMerge (master)
$ git push -u origin master
Username for 'https://github.com': odeb
Counting objects: 22, done.
Delta compression using up to 6 threads.
Compressing objects: 100% (18/18), done.
Writing objects: 100% (22/22), 2.02 KiB | 0 bytes/s, done.
Total 22 (delta 7), reused 0 (delta 0)
remote: Resolving deltas: 100% (7/7), done.
To https://github.com/odebb/sioCrawler.git
 * [new branch] master -> master
Branch master set up to track remote branch master from origin.
```

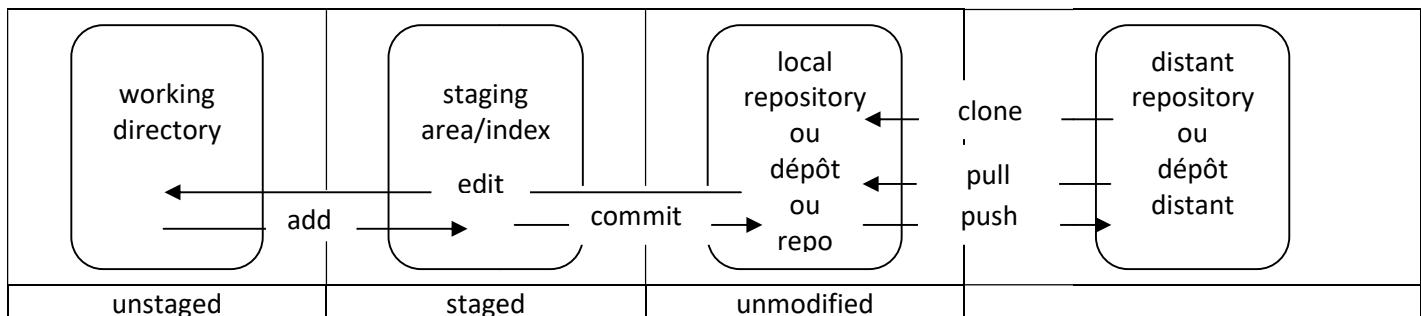


éventuellement si vous avez des messages d'erreur liés à ssl vous aurez à configurer le chemin du certificat. Le chemin et le nom du fichier .crt peut changer en fonction de la configuration du poste

```
MINGW64:/e/git/devMerge
odeb@odeb-ws MINGW64 /e/git/devMerge (master)
$ git config --system http.sslcainfo "C:\Program Files\Git\mingw64\ssl\certs\ca-bundle.crt"
```

5. Commandes spécifiques à un dépôt distant

Tout ce que vous avez fait pour manipuler git reste valable, mais nous venons de rajouter un élément supplémentaire : un dépôt distant qui servira à centraliser les ou les projets de plusieurs développeurs.



Après avoir commité vous pouvez transmettre au dépôt distant vos modifications :

`git push origin master`

```

MINGW64:/e/git/sioCrawler
odeb@odeb-ws MINGW64 /e/git/sioCrawler (master)
$ git remote -v
origin https://github.com/odebb/sioCrawler.git (fetch)
origin https://github.com/odebb/sioCrawler.git (push)

odeb@odeb-ws MINGW64 /e/git/sioCrawler (master)
$ git push origin master
Username for 'https://github.com': odeb
Everything up-to-date

```

Attention le dépôt distant est commun à plusieurs développeurs, il faut donc impérativement se tenir au courants des modifications faites. Il faut le faire assez régulièrement sinon vous risquez de développer sur du code obsolète ou de créer des conflits de plus en plus complexes à résoudre :

`git pull origin master`

```

MINGW64:/e/git/sioCrawler
odeb@odeb-ws MINGW64 /e/git/sioCrawler (master)
$ git pull origin master
From https://github.com/odebb/sioCrawler
 * branch            master      -> FETCH_HEAD
Already up-to-date.

```

Gestion des conflits

Les conflits se résolvent de la même manière qu'en local jusqu'à ce que l'on puisse effectuer un push correct.

Enfin si vous voulez commencer à travailler en local en récupérant l'ensemble d'un dépôt la commande clone permet de créer un dépôt local à partir d'un dépôt distant. Attention elle est plus consommatrice de ressources que la commande pull qui elle ne récupère que les nouveautés.

`git clone`
`https://github.com/odebb/sioCrawler.git`

```

MINGW64:/e/git
odeb@odeb-ws MINGW64 /e/git
$ git clone https://github.com/odebb/sioCrawler.git
Cloning into 'sioCrawler'...
remote: Counting objects: 22, done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 22 (delta 7), reused 22 (delta 7), pack-reused 0
Unpacking objects: 100% (22/22), done.

```

Vous connaissez maintenant les commandes :

- ✓ `git clone`,
- ✓ `git pull`
- ✓ `git push`

6. Documenter un développement, paramétrage supplémentaire

Pour chaque dépôt vous pouvez ajouter trois fichiers particuliers :

- le fichier readme.md,
- le fichier .gitignore,
- un fichier comportant la licence d'utilisation de votre développement.

6.1 Le fichier readme.md

Chaque dépôt doit être documenté c'est le fichier readme.md qui contiendra la documentation du développement ainsi que la manière dont on peut y participer si c'est un développement communautaire.

L'extension .md signifie que c'est un fichier MarkDown qui est un langage à balise au même titre que le html. En fait c'est une simplification du html on retrouve dans MarkDown toutes les balises de mise en forme d'un texte en html.

Pour chaque projet ce fichier est à documenter. Pour vous c'est l'élément essentiel de votre Portfolio.

Chaque activité du Portfolio doit correspondre à un dépôt : les sources du développement + le fichier ReadMe.md qui décrit votre activité de développement.

Pour chaque activité donc dépôt on doit retrouver une description illustrée par :

- un diagramme acteur flux décrivant l'activité de l'entreprise ou de la fonctionnalité développé,
- des cas d'utilisation décrivant chacune des fonctionnalités du développement,
- un modèle conceptuel décrivant l'organisation des données,
- un diagramme de classe,
- l'arborescence des menus ou fonctionnalités de l'application,
- des exemples de maquette écran,
- un planning de développement,
- diagramme de séquence et ou diagramme état/transition et ou diagramme

Voici un exemple de fichier readme.md : essayez de le reproduire, on vous fournit les images écran de tous les images :

Screenshot of a GitHub repository page for "sioCrawler". The page shows 12 commits, 1 branch, 0 releases, and 1 contributor. The branch is "master". A pull request titled "sioCrawler résolution conflit merge" has been merged. The commit history includes:

- images ajout du README.md ainsi que du repertoire images pour les illustrations... 23 minutes ago
- Arme.cs Creation de la classe Arme 2 days ago
- Heros.cs commit initial 3 days ago
- Monstre.cs Creation de la classe Monstre 3 days ago
- Personnage.cs résolution conflit merge sio/master sur classe Personnage 4 days ago
- readme.md résolution conflit merge 5 minutes ago

The repository contains a single file: "readme.md".

projet sioCrawler

présentation du projet : le but est de réaliser un jeu de rôle permettant un jeu peu mûrement multi joueur permettant aux étudiants du bts sio de se divertir pendant les heures de cantine mais surtout d'améliorer leurs compétences en développement.

Les outils mis en œuvre :

- git,
- Visual studio,
- mysql,
- apache.

Le développement tourne autour de 3 grandes parties

- 1 inscription en ligne
- 2 développement du jeu en lui-même permettant l'exploration d'un labyrinthe
- 3 la sauvegarde des personnages et du contexte du jeu

développement	langages	technique de programmation
inscription en ligne	php, MySQL	développement web MVC avec Code Igniter
sio crawler le jeu	c#	programmation objet, tests unitaires
sauvegarde du contexte	c#, mysql	programmation procédural procédures stockées en mysql

L'inscription en ligne.

site web permettant à un joueur de s'inscrire en ligne. le projet prévoit le principe suivant pour l'inscription en ligne.

```
sequenceDiagram
    participant Player
    participant Site
    participant Group
    Player->>Site: inscription
    activate Site
    Note over Site: 1 inscription
    Site-->>Player: confirmation d'inscription
    activate Player
    Note over Player: 2 demande de confirmation d'inscription
    Player-->>Group: rejoindre
    activate Group
    Note over Group: 3 confirmation
    Group-->>Player: nouvelle Partie
    activate Player
    Note over Player: 4 rejoindre
    Player-->>Group: 5 démarer
    deactivate Player
    deactivate Group
    deactivate Site
```

sio crawler le jeu

le joueur possédera les fonctionnalités suivantes.

```
actor Player
    actor SeDeplacer
    actor Combatir
    actor Explorar
    Player --> SeDeplacer
    SeDeplacer --> Combatir
    SeDeplacer --> Explorar
    Combatir --> Explorar
```

les classes développées.

```
classDiagram
    class Personnage {
        <<Frapper(Personnage personnage, void entForce)>>
        <<Deplacer(Direction direction, void Lieu)>>
        <<Reacter(Direction direction, int distance)>>
    }
    class Arme {
        <<Object id description points>>
    }
    class Monstre
    class Heros
    Personnage "0..1" --> "0..1" Arme
    Personnage "0..1" --> "0..1" Monstre
    Personnage "0..1" --> "0..1" Heros
```

Sauvegarde du contexte

La sauvegarde du contexte se fera dans la base de données.

```
classDiagram
    class Personnage {
        <<Id nom pourcentage de force intelligence force>>
    }
    class Object {
        <<Id description points>>
    }
    class Context
    Personnage "0..n" --> "0..n" posseder
    Personnage "0..n" --> "0..n" porter
    Object "0..n" --> "0..n" posseder
    Object "0..n" --> "0..n" porter
    Context "0..1" --> "0..1" porter
```

Schémas à intégrer dans le document.

- acteur flux : acteurFluxInscription.png
- cas d'utilisation fonctionnalités à développer : useCasePersonnage.PNG
- mcd sauvegarde du jeu : mcdSauvegarde.PNG
- diagramme de classe du développement : diagrammeClassePersonnage.PNG

6.2 Le langage MarkDown

Voir annexe.

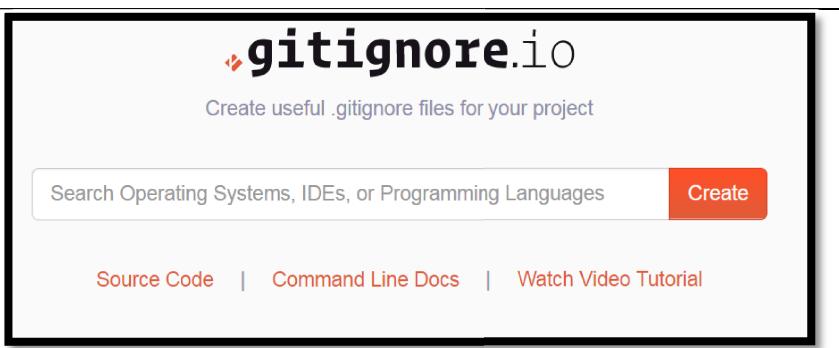
6.3. Ignorer le suivi de certains fichiers : le fichier .gitignore

.gitignore est un fichier qui contient la liste des fichiers et/ou des répertoires dont on ne doit pas tenir compte lors de toutes les commandes git.

Par exemple si vous utilisez Visual Studio pour développer, celui-ci va générer un grand nombre de fichiers .exe, .dll etc que nous ne voulons pas suivre lors du développement. En gros ce qui nous importe ce sont les fichiers .cs, le .sln et le .csproj les autres fichiers sont générés lors de la phase de compilation.

Certains sites permettent de générer des fichiers .gitignore par défaut, en fonction du langage : php, python, c sharp etc.

Allez sur ce site <https://www.gitignore.io> générez un fichier .gitignore pour un développement cs en Visual Studio et intégrez le dans votre développement.



7. Travailler à plusieurs en réseau sur un dépôt commun

Cette partie doit être réalisé à plusieurs (au moins 2 ou 3). La commande clone ne doit être utilisée que dans la question a).

scénario

a) Création d'un dépôt commun

- ⇒ user1 : initialise le projet précédent sur gitHub
- ⇒ user2 : clone le projet de user 1
- ⇒ user3 : clone le projet de user1

b) Chaque développeur travaille sur sa partie-

- ⇒ user1 : ajoute une nouvelle classe
- ⇒ user2 : ajoute une nouvelle classe
- ⇒ user3 : ajoute une nouvelle classe
- ⇒ Chaque utilisateur propage en local ses modifications

c) Il est temps faire profiter les autres de son travail

- ⇒ user3 a fini son travail, il push son travail sur gitHub
- ⇒ user1 et user2 vont récupérer ce qu'il a fait par un pull

d) Tout le monde doit récupérer le travail des autres.

- ⇒ user1 push son travail
- ⇒ user 2 push son travail
- ⇒ tout le monde doit récupérer le travail des autres.

e) Tous les développeurs travaille sur une même partie

- ⇒ mettez vous d'accord sur une class à modifier, chaque développeur ajoute une nouvelle méthode à cette class
- ⇒ chaque développeur push ses modifications
- ⇒ à la fin tout le monde doit récupérer un projet complet.

8. les autres commandes utiles

références relatives git		
git cherry -pick		
git reset	<pre>git reset HEAD nomFichier</pre> <pre>git reset --hard HEAD nomFichier</pre> <pre>git reset nomFichier</pre>	annuler un git add avant de lancer le commit
git revert		
gérer des tag	git tag v0.9	
git stash		
git commit --amend		
git blame		
git branch --f [origine] [destination]		modifier l'emplacement d'une étiquette de branche
git pull	<pre>git fetch</pre> <pre>git merge FETCH_HEAD</pre>	décomposition d'un git pull
git pull --rebase		pour effectuer un rebase au lieu d'un merge
git rm source.cs ou rm fichier git add source.cs		
git diff	<pre>git diff</pre> <pre>git diff --cached</pre> <pre>git diff HEAD</pre>	working dir != index index != local repository working dir != local repository
workflow	master hotfix develop release feature(local)	
références	^	HEAD^2 : 2 commits après HEAD
	~	inverse de l'autre