GIT

In case of fire

1. git commit

2. git push

3. leave building

C koi ?

Un logiciel de gestion de version décentralisé

Utilisable en ligne de commande ou visuellement

Difficile à maîtriser dans les cas spécifiques mais indispensable à tout projet

Développé par Linus Torvald, aka le papa de Linux

POURQUOI?

ATTENTION

Installation

Sur Windows: installer Git Bash:

https://gitforwindows.org/

Sur Linux normalement il est installé par défaut sinon:

sudo apt install git

etc en fonction de votre distribution

Fonctionnement de base

On commence toujours par:

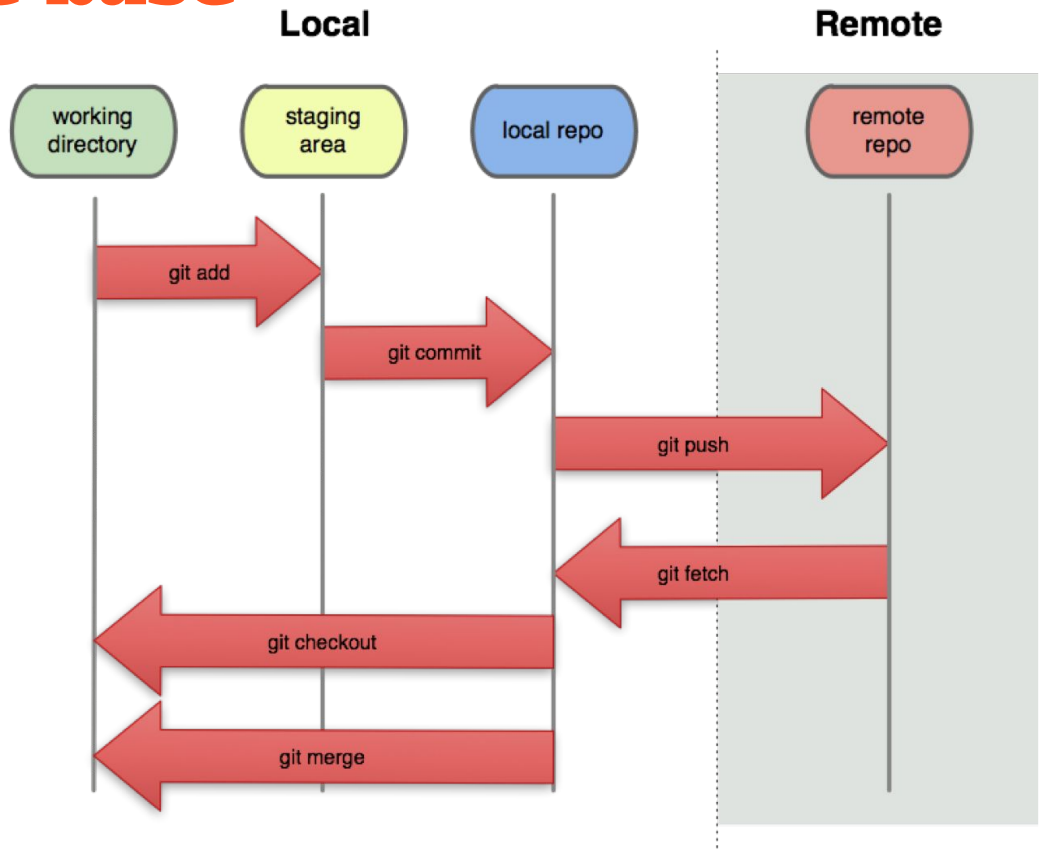
En local on crée son projet:

```
git init
```

ou

En reprenant un projet distant:

```
git clone <url_repo>
```



Fonctionnement de base

On ajoute les fichiers:

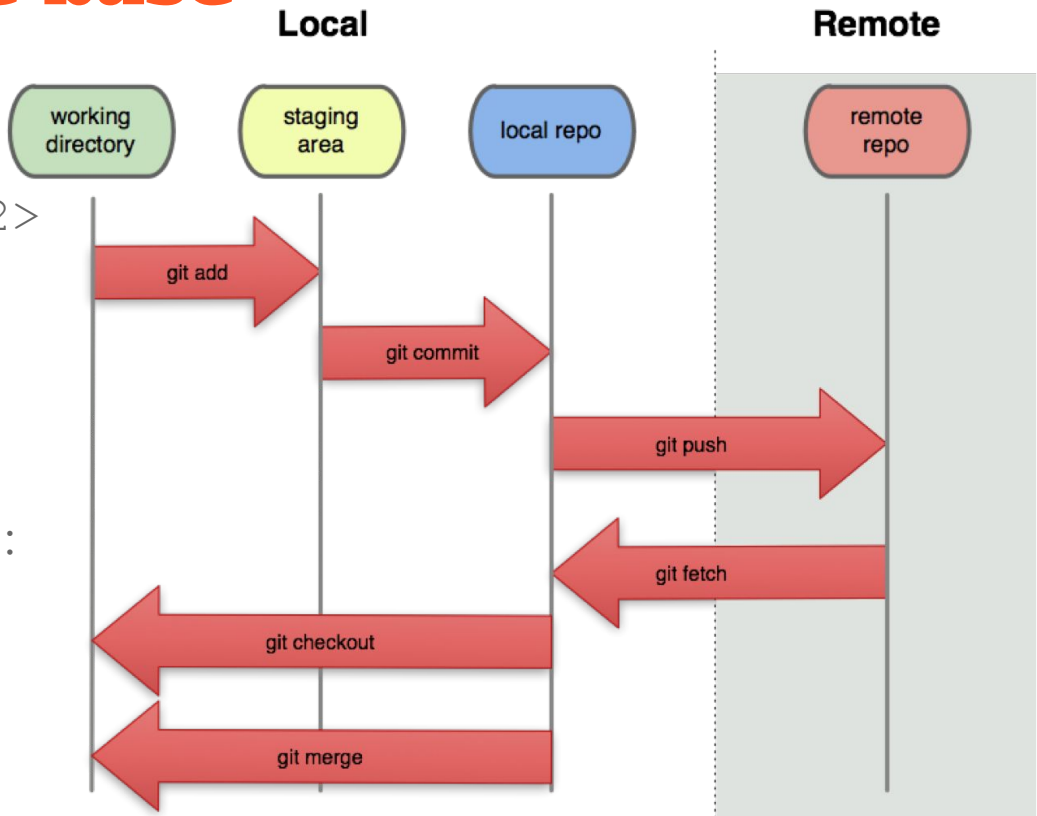
```
git add <fichier1> <fichier2>
```

plus généralement :

```
git add *
```

puis on peut vérifier les fichiers ajoués :

```
git status
```



au cas où : `git rm` et `git mv`

Fonctionnement de base

On peut alors enregistrer les changements :

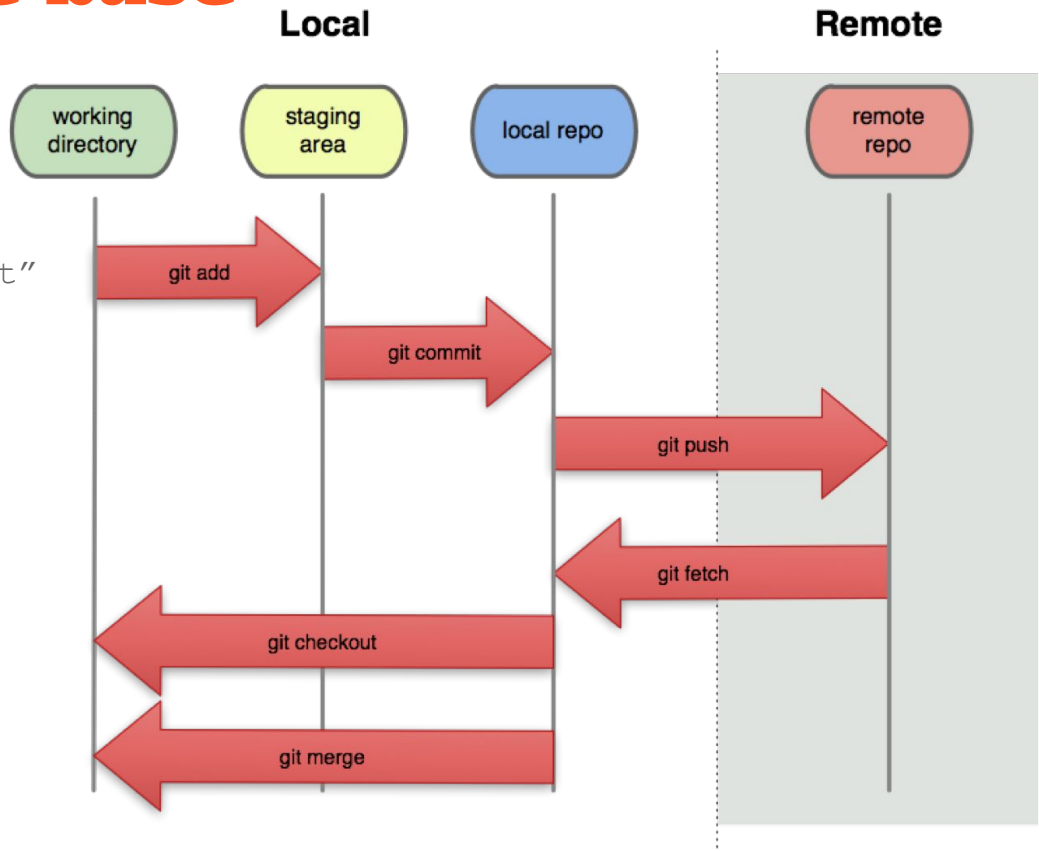
```
git commit -m "qu'est-ce que j'ai fait"
```

puis les envoyer en remote :

```
git push
```

Et aller chercher les modifications :

```
git pull
```



Les branches

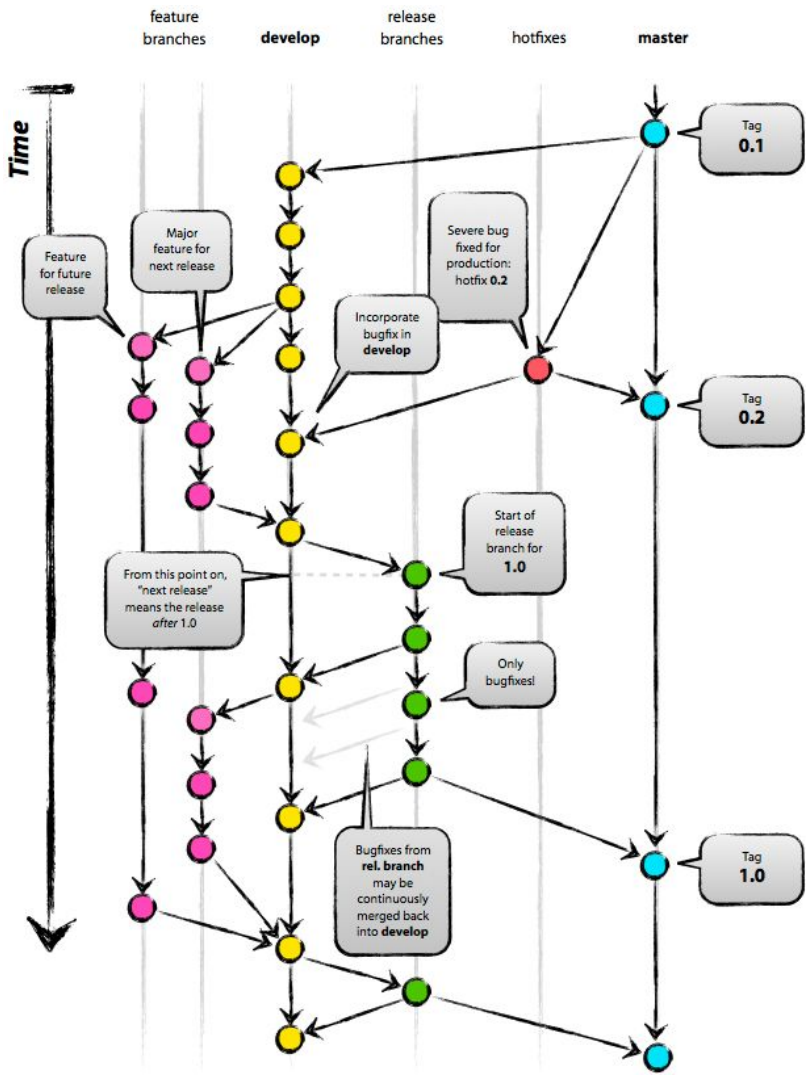
Objectif: mettre de l'ordre dans son projet

Commandes:

Créer : `git branch <nom_branche>`

ou `git checkout -b <nom_branche>`

Changer : `git checkout <nom_branche>`



Ça se complique...

Naviguer entre différentes versions

```
git checkout <branche/id commit/HEAD...>

git diff <commit1> <commit2>
```

HEAD : dernier commit sur la branche actuelle

HEAD^ : avant-dernier commit

HEAD^^ : antépénultième commit

HEAD~2 : antépénultième commit

donc HEAD~4 = HEAD^^^^

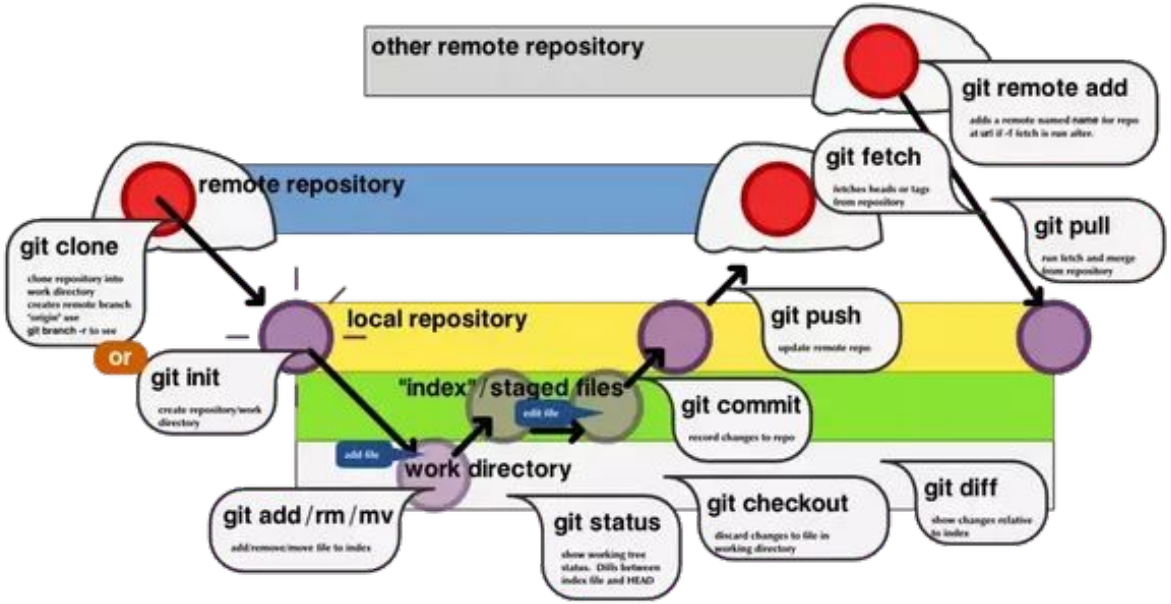
Commandes git

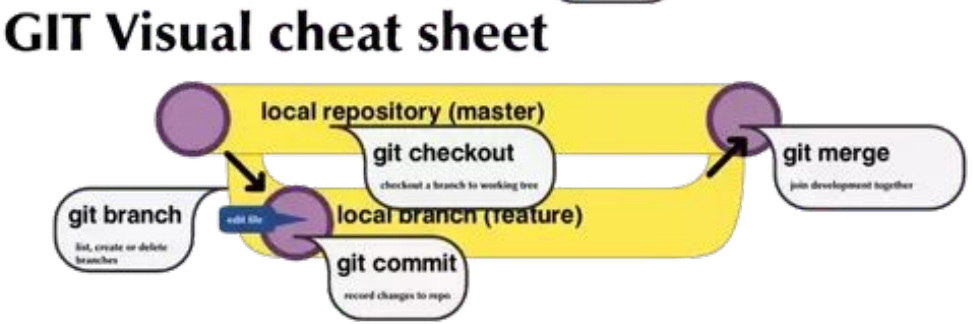
La base

- git init
- git clone
- git add <files>
- git rm <files>
- git commit -am “le message”
- git pull
- git push
- git branch <branch>
- git checkout <branch>
- git log
- git status

La difficulté

- git merge <branch>
- git rebase <branch>
- git stash





git merge

Deux branches à joindre ? trop facile !

se positionner sur la branche principale puis :

```
git merge <branche_à_merge>
```

Bravo ! vous avez fait la fusion de deux branches

Si on veut mettre ses changements de côté sans commit, ou si on veut pull alors qu'on a des changements :

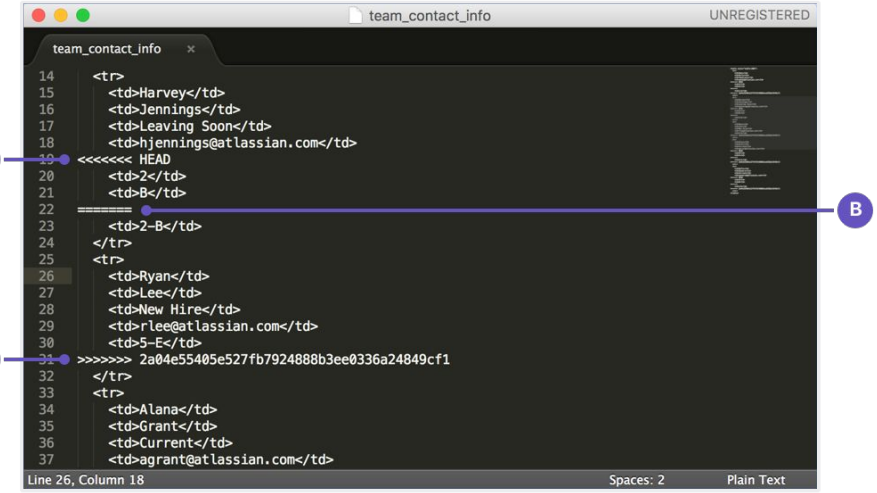
```
git stash

git stash apply
```

Les merge conflicts

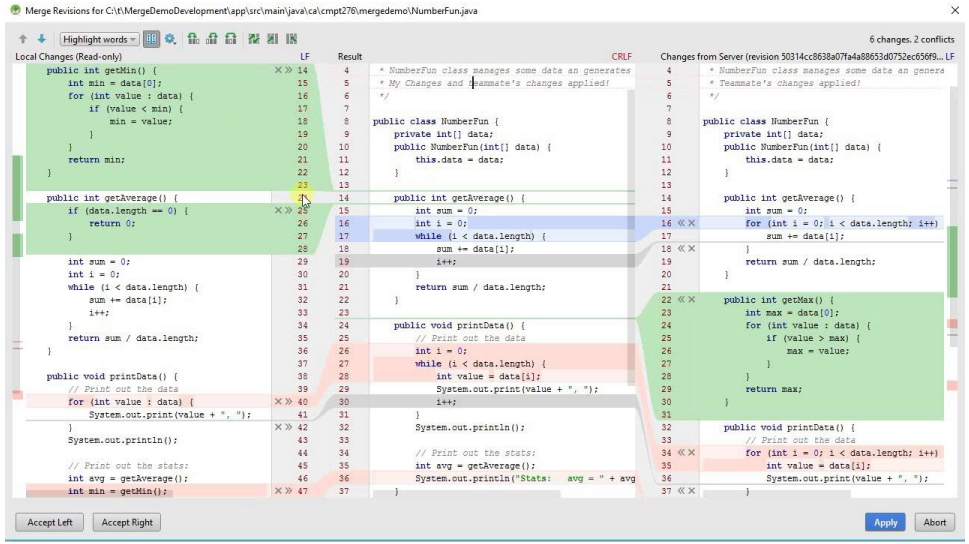
Arrivent quand plusieurs personnes modifient différemment et en parallèle un même fichier

Plusieurs manière de les résoudre:

Manuellement



Depuis son IDE



Se connecter à Github/Gitlab/Bitbucket

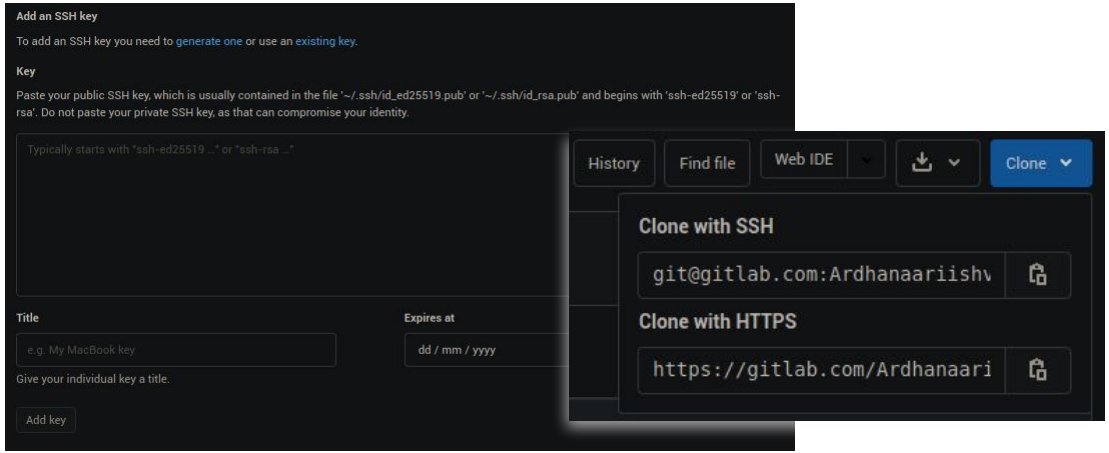
Pour clone ou push, vous devrez vous créer un compte sur une de ces plateformes.

Par défaut, chaque push/clone (**HTTPS**) demande votre mot de passe

Authentification SSH : paire de clé (public, private) à générer

communiquer votre clé publique dans les paramètres de votre compte GitHub/GitLab/Bitbucket

```
klar (11:39) ~>ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ylo/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ylo/.ssh/id_rsa.
Your public key has been saved in /home/ylo/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:Up6KjbnEV4Hgfo75YM393QdQsK3Z0aTNBz0DoirrW+c ylo@klar
The key's randomart image is:
+---[RSA 2048]----+
|    .      ..oo..|
|   . . .  . .o.X.|
|    . . o.  ..+ B|
|   .   o.o  .+ ..|
|    ..o.S   o..  |
|   . %o=      .  |
|    @.B...     . |
|   o.=. o. . .  .|
|    .oo  E. . .. |
+----[SHA256]-----+
klar (11:40) ~>
```



.gitignore

Liste les fichiers et dossiers qui seront ignorés par Git

ex: caches (inutile), fichiers confidentiels (sécurité), fichiers de configuration de l'IDE (inutile)

Syntaxe très simple :

- noms relatifs des fichiers/répertoires à ignorer
- / à la fin match uniquement des répertoires
- # permet de commenter
- ! prend la négation
- * match tout sauf \
- ? match un unique caractère sauf \
- ** spéciale :
    - <truc>/** -> tous les fichiers dans truc
    - **/<truc> -> englobe tous les répertoires menant à un truc
    - <machin>/**/<truc> -> englobe 0 ou plus répertoires entre machin et truc

https://gitignore.io : génère des .gitignore

```
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
wheels/
share/python-wheels/
*.egg-info/
.installed.cfg
*.egg
MANIFEST

# PyInstaller
#  Usually these files are written by a python script from a template
#  before PyInstaller builds the exe, so as to inject date/other infos into it.
*.manifest
*.spec

# Installer logs
pip-log.txt
pip-delete-this-directory.txt

# Unit test / coverage reports
htmlcov/
.tox/
.nox/
.coverage
.coverage.*
.cache
nosetests.xml
coverage.xml
```

Concrètement

On ne git add pas tous les fichiers d'un projet ! ⟶ on fait un **.gitignore**

On utilise les intégrations git de son IDE par exemple pour gérer les merge conflicts

On crée des branches (intelligemment)

Conventions utiles:

- **README.md**
- requirements

Apprendre en pratiquant

Les utilisations des branches visuellement: https://learngitbranching.js.org/

Les commandes de git en général: https://gitexercises.fracz.com/

Autres ressources

Un super site résumant les commandes utiles en cas de scénario terrible:

https://ohshitgit.com/

Le résumé des commandes essentielles:

https://education.github.com/git-cheat-sheet-education.pdf

Guide de survie en remote

Merge requests

Fork

Issues

CI/CD

License

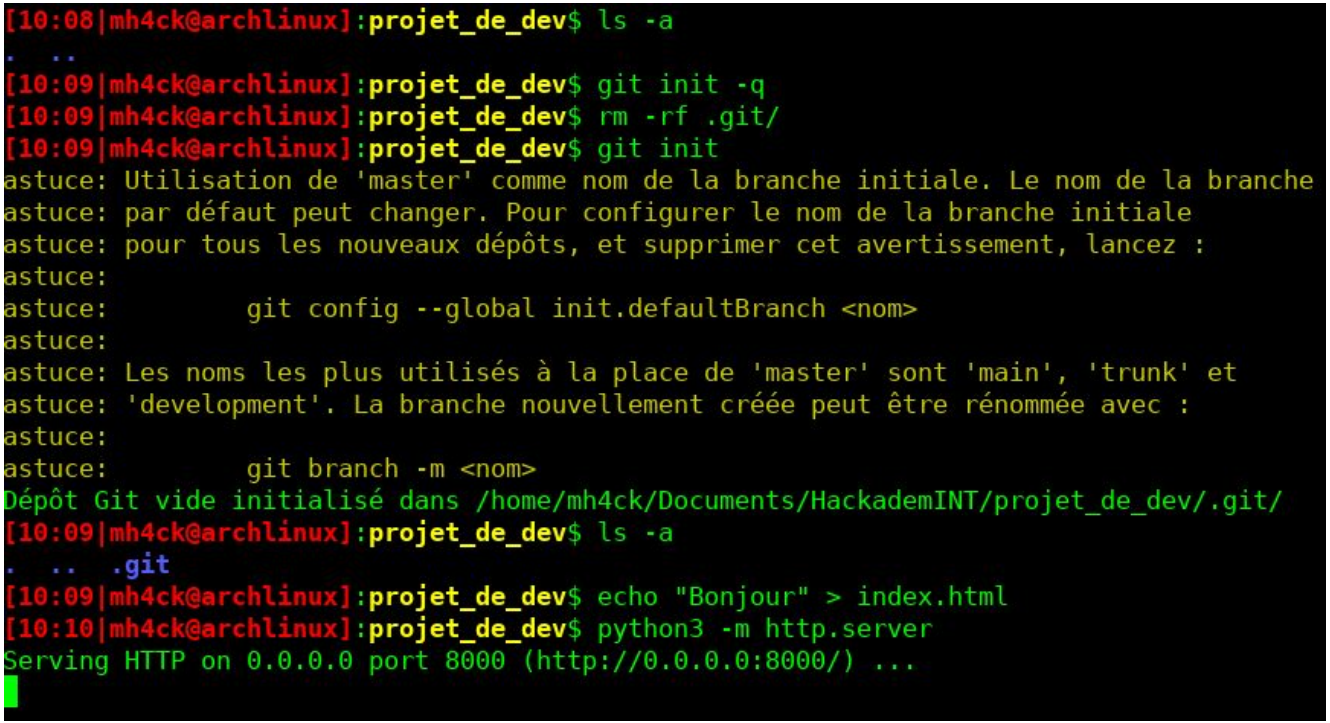
L'instant sécu avec HackademINT

Comme avec n'importe quoi dans le monde de l'informatique, une mauvaise utilisation de git peut mener à des vulnérabilités !

- Vulnérabilité de l'exécutable git
- Vulnérabilité des hébergeurs (github, gitlab, gitea...)
- Mauvaises utilisations de git menant à des vulnérabilités

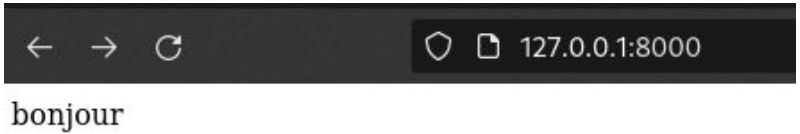
Ne PAS développer en production

git stocke toutes les informations nécessaires à son fonctionnement dans un dossier caché .git ...



Ne PAS développer en production

... ce qui peut mener à de grosses vulnérabilités si ce dossier est rendu accessible !!



PAS BIEN : il est possible de récupérer l'intégralité du repo, dont le code source, tous les commits, toutes les branches...!

Réfléchir au contenu des commits

- Fichiers de configuration contenant des clés ou des mots de passe rentrés à la main (config.php, config.xml, .env, .config ...)
- Fichiers de configuration de plugins ou d'extensions avec des clés générées aléatoirement et stockées dans votre projet (CMS, extensions PHP, extensions VSCode...) et aussi des secrets (tokens) GitHub, GitLab... pouvant être stockés dans votre projet et commit par erreur
- Clés personnelles (id_rsa, password.txt, MDPsDeLaBanque.kdb ...)
- Attention aux commentaires avec des infos sensibles dedans : ils sont aussi lisibles si on récupère le repo
- Je Fais De La SeCu CaR jE CoMmIt PaS dEs FiChIeRs SeNsIbLeS mais je file des informations confidentielles dans les noms des commits ("astuce" utilisée par de nombreuses personnes) : NE FONCTIONNE PAS ! Avec la vulnérabilité présentée juste avant, on récupère même le nom et les messages des commits

=> De manière générale, on réfléchit avant de faire un git add * !!

C'était l'instant sécu avec HackademINT

Des questions ?

Bon TP à tous / toutes, et n'oubliez pas de réfléchir à ce que vous faites ;-)