Versionnage avec Git

Gérard Rozsavolgyi

roza@univ-orleans.fr

Jan 2022

GitHub

Code Collaboration & Version Control

Bitbucket

Code Collaboration & Version Control

roza@univ-orleans.fr

Les systèmes de gestion de version

Git

Formations 2022

- Plan:
  - Notions générales sur les systèmes de gestion de version
  - Git, prise en main
  - Les branches en Git
  - Introduction aux tests

Fiches memo sur Git en pdf

- Git commandes de base
- Git collaboratif dépôt unique
- Git collaboratif dépôts distincts
- Git collaboration avec des branches sur dépôt unique
- Git collaboration avec des branches et dépôts distincts
- Git merge et rebase

Slides Git en pdf

- Ces slides en pdf (4/3)
- Ces slides en pdf (16/9)
- Ces slides en pdf (16/10)

Problématique 1

conserver l'historique des modifications apportées chaque jour … et revenir en arrière en cas de besoin

Problématique 2

- partager le développement entre plusieurs personnes
- permettre un accès partagé au système de fichiers versionné
- permettre un accès distant depuis diverses plateformes
- retrouver rapidement l'origine d'un Bug en cas de régression

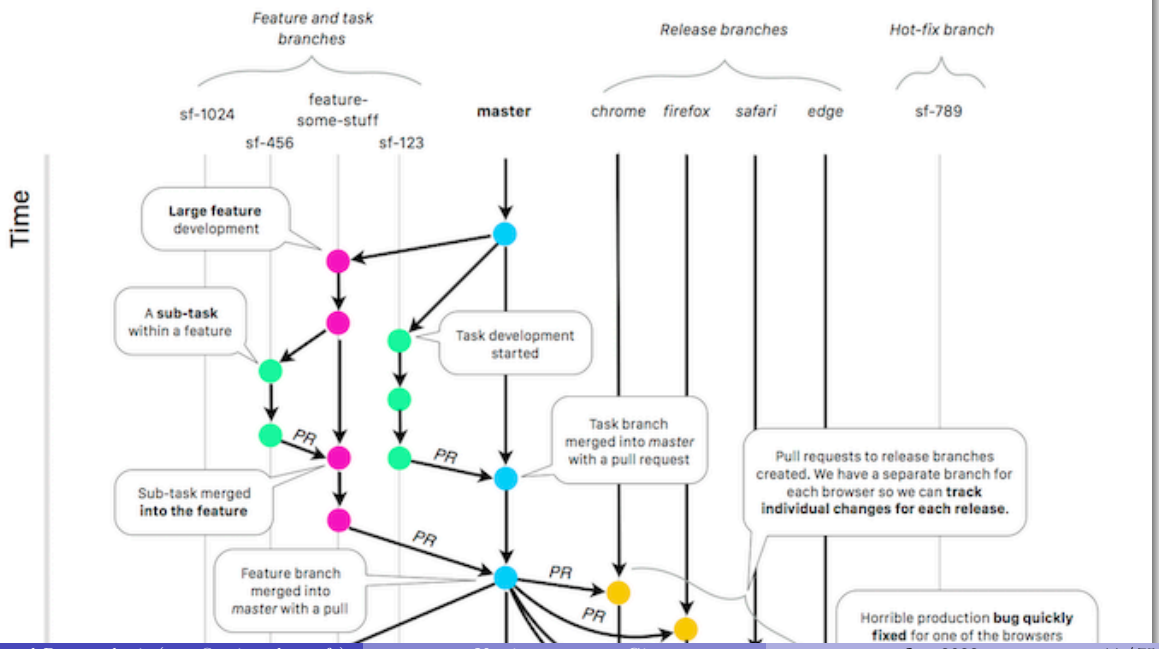
Problématique 3

Comment gérer des distributions ?
↪ plusieurs versions peuvent coexister (branches de développement, branches beta, branches production)

Problématique 4

Utiliser de l'intégration continue (Gitlab, Github, Jenkins, etc.)
↪ introduire des vérifications, tests, production d'artefacts lors d'une mise à jour d'un dépôt.

Complexe


Feature and task branches
Release branches
Hot-fix branch
sf-1024
feature-some-stuff
sf-456
sf-123
master
chrome
firefox
safari
edge
sf-789
Time
Large feature development
A sub-task within a feature
Task development started
PR
Task branch merged into master with a pull request
Sub-task merged into the feature
Pull requests to release branches created. We have a separate branch for each browser so we can track individual changes for each release.
Feature branch merged into master with a pull
Horrible production bug quickly fixed for one of the browsers

Ou

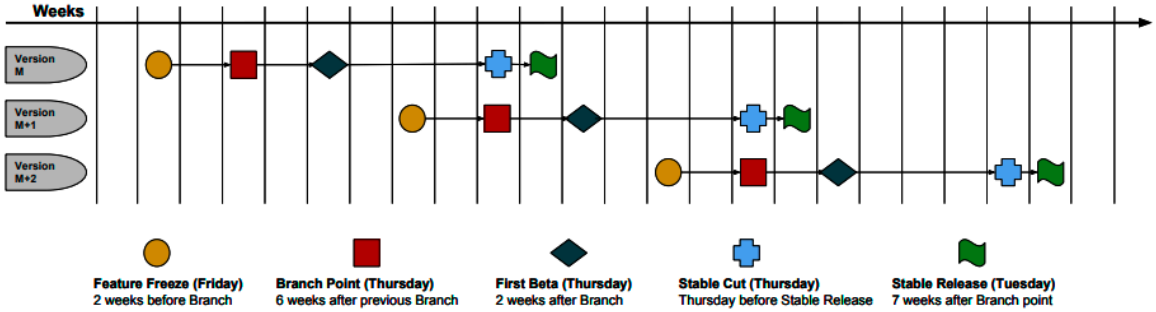


Figure 2: Branches de chrome…

Un logiciel de gestion de version

- Agit sur une arborescence de fichiers
- Permet de mutualiser un développement
- Permet de stocker toute évolution du code source

↪ Ajoute une nouvelle dimension au système de fichiers: le temps

2 modèles

- Centralisés
- Décentralisés

Centralisés

- Style CVS ou SVN
- un seul dépôt de référence
- besoin d'un serveur
- besoin de synchronisation
- conflits plus fréquents ...

Décentralisés

- plusieurs dépôts pour un même logiciel
- chacun peut travailler à son rythme
- de façon synchronisée ou pas des autres

Systèmes décentralisés

- Bazaar
- Mercurial
- Git

Avantages

- ne pas être dépendant d'une machine
- travailler sans connexion
- participation "progressive" à un projet:
  - accès au code
  - contribution proposée
  - contributeur "actif" si acceptée

Dépôt de référence

- Dépôt contenant les versions livrées d'un projet
- Un dépôt est un emplacement central où sont stockées :
  - l'historique des versions des fichiers
  - les logs
  - les dates, auteurs, tags, etc.

Dépôt

Un dépôt apparaît de l'extérieur comme un système de fichiers composé de répertoires au sein desquels on peut naviguer, lire et écrire selon les permissions dont on dispose.

Master (ou Trunck en SVN)

- Version principale
- à partir d'elle, on peut crééer des branches

Branches

- un développement « secondaire » est mis en route
- nouvelle fonctionnalité
- correction de bugs, etc.

Destin d'une branche

- Une branche peut soit être à nouveau fusionnée dans le « master »
- soit disparaître
- soit donner lieu à un nouveau programme. On parle alors de fork

Le Checkout

- consiste à récupérer pour la première fois les fichiers déjà existant au sein d'un projet du dépôt
- Le résultat est une copie de travail
- Sous git cela consiste aussi à choisir sa branche de travail

Ajout

Pour ajouter un fichier spécifique:

```
git add monfic
```

Pour ajouter tout le contenu du répertoire courant:

```
git add .
```

Commit

- Met à jour la copie locale (puis si on veut, le dépôt)
- Une nouvelle révision est alors créée
- il faut que la copie de travail corresponde à la dernière version du dépôt
- un message est associé au commit
- éventuellement un tag

Commit

```
git commit -m "added a wonderful new feature"
```

ou si on veut ajouter en même temps les modifications portant sur des fichiers existants :

```
git commit -am "added a wonderful new feature"
```

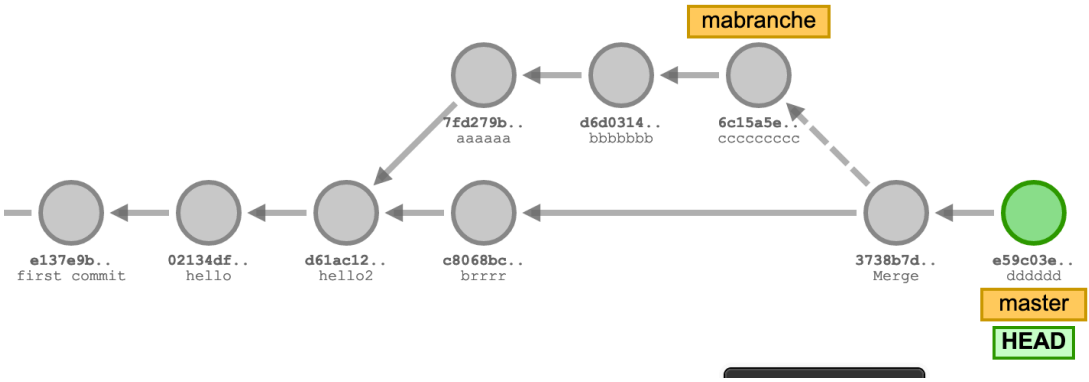
update (pull)

- L'update synchronise la copie de travail locale avec le dépôt
- en récupérant la dernière version des fichiers du dépôt
- C'est à cette occasion que des conflits de version peuvent apparaître :)

Merge

- Fusion de branches entre elles
- Fusion d'une branche avec le master (main)

Schéma



git commits et branches

Conflits

- certaines modifications peuvent être contradictoires
- par exemple lorsque deux personnes ont apporté des modifications différentes à la même partie d'un fichier ...
- il faut alors résoudre ce conflit !

Diff

- Permet d'afficher les différence entre deux versions d'un fichier
- kdiff pour linux ou autres

mergetools

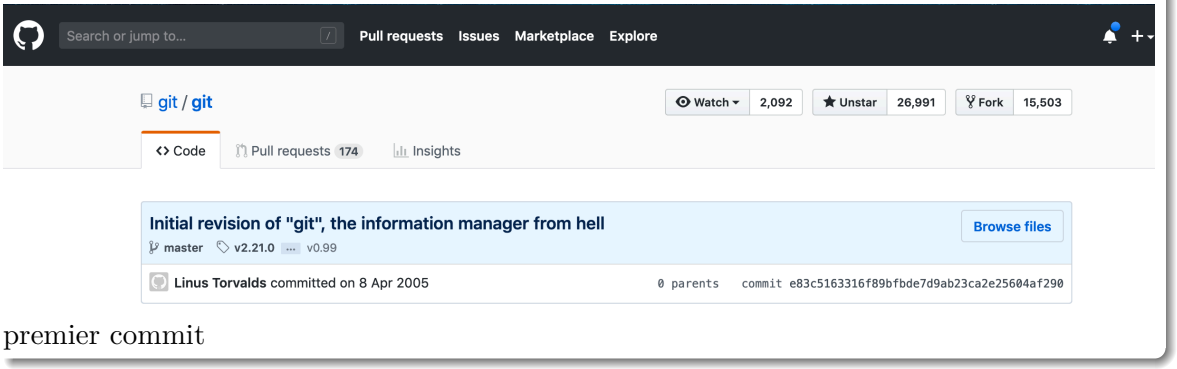
- vscode
- atom
- vim
- kdiff3
- opendiff

Git

- Le plus utilisé
- créé par Linus Torvalds pour gérer le noyau de Linux
- centaines de contributeurs

Historique video Linux Kernel contributions

Première révision par Linus T.



premier commit


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

commandes de base

```
alan> git config --global user.name "Alan Turing "
alan> git config --global user.email "alan@univ-orleans.fr"
alan> git config --global core.editor emacs
alan> git config -l
```

(choisir votre editeur)

git help

Documentation

```
bob> git --help
bob> git help -a
bob> git help init
```

travailler avec git

```
git init
```

- un dossier .git est créé
- Pour voir son contenu :

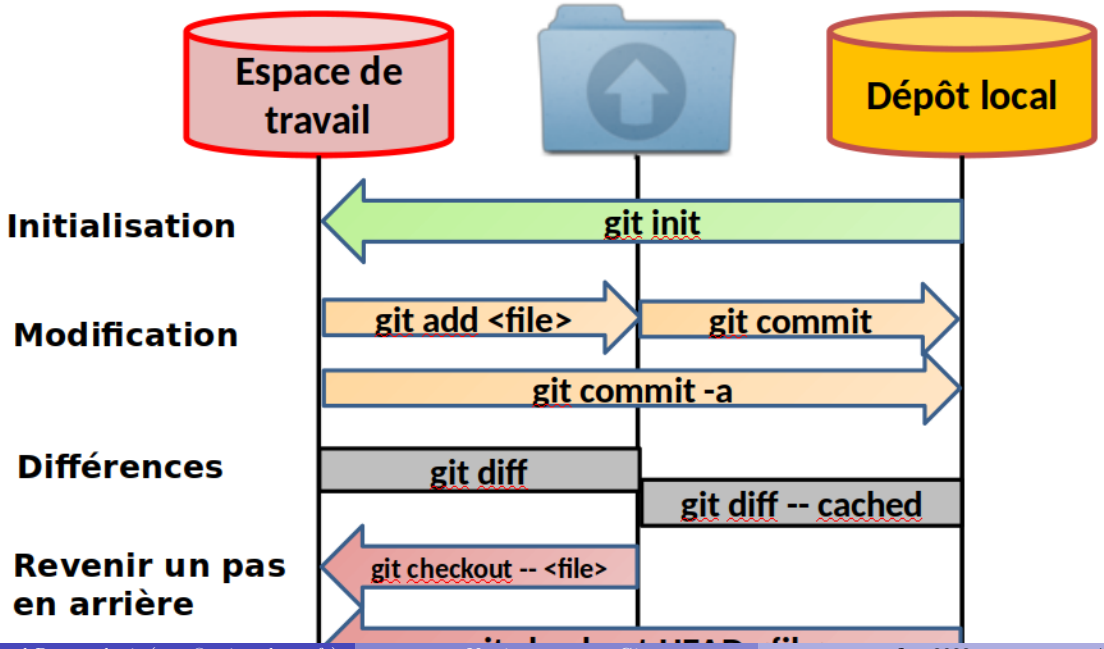
```
tree .git
```

Ajoutons un fichier Readme

```
git add Readme.md
```

Readme.md est à présent dans l' index mais pas encore commité … AJoutons-le:

```
git commit -m "ajout Readme.md"
```


Espace de travail
Dépôt local
Initialisation
git init
Modification
git add <file>
git commit
git commit -a
Différences
git diff
git diff -- cached
Revenir un pas en arrière
git checkout -- <file>

un commit

- operation atomique pour le dépôt git
- pas trop gros, pas trop petit
- contribution unique
- ne pas réparer 3 bugs et ajouter 2 nouvelles fonctionnalités dans le même commit !!

rm

- N'effacez pas un fichier versionné directement
- Faites-le sous le contrôle de git
- Idem pour un renommage

```
git rm monfichier
```

Premier travail

- créer un petit projet demo
- y ajouter un Readme.md
- commiter
- changer quelque chose au Readme.md
- utiliser `git diff` pour afficher les différences
- commiter la nouvelle version
- N'oubliez pas d'utiliser `git status`, `git log`, etc.

Configurer un dépôt distant

- Une fois que vous avez un contenu versionné localement
- crééez un dépôt distant, par exemple: `https://gitlab.com/bob/demo.git`
- pour publier votre travail :

```
git remote add origin https://gitlab.com/bob/demo.git
```

publiez votre travail

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

et quand vous aurez du nouveau contenu à mettre à jour :

```
git push
```

git status

Pour vérifier :

- Si vous avez du nouveau contenu à commiter
- sur quelle branche vous travaillez
- S'il vous manque du contenu du dépôt distant
- ou l'inverse : `Your branch is ahead of 'origin/master' by 3 commits.`

update (pull)

`git pull`

- Synchronise le dépôt local avec le distant
- A cette occasion, des conflits peuvent apparaître ...

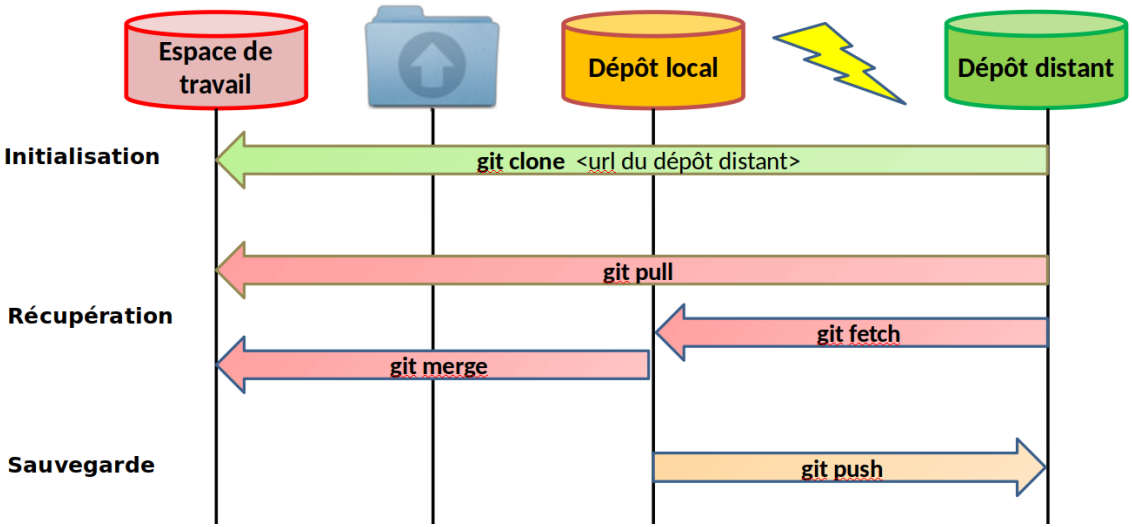


Figure 5: git distant

Où entreposer votre code

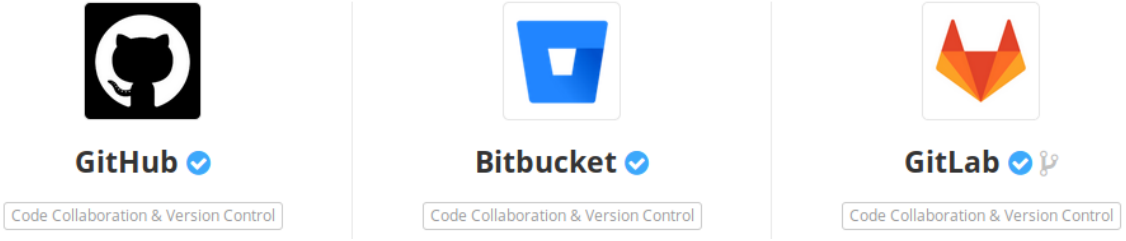


Figure 6: Github, Gitlab, etc.

Les entrepôts de code connus

- Github permet maintenant des dépôts privés gratuits depuis le rachat par Microsoft... Voir
- Gitlab et bitbucket le permettaient déjà
- Vous pouvez télécharger et installer votre propre gitlab ou Gogs

ignorer certains fichiers

- ajouter et commiter un nouveau fichier .gitignore
- contenant une liste de fichiers ou extensions que vous ne voulez pas versionner. Comme par exemple :

```
*.o
__pycache__
*.class
*~
```

Ajouter d'autres remotes

Si vous voulez travailler avec Alice:

```
git remote add alice https://gitlab.com/alice/monproject.git
```

lister vos remotes:

```
git remote -v
```

Créer et utiliser une branche

```
git branch db-integration
git checkout db-integration
```

en une ligne :

```
git checkout -b db-integration
```

Contribuer à un projet opensource

1. Choisir un projet sympa (nodejs, ruby on rails, d3js, etc.) sur github ou gitlab
2. forkez-le sur Github ou Gitlab
3. clonez votre fork localement
4. choisissez un bug signalé ou une nouvelle fonctionnalité demandée Par exemple NodeJS Github repo

Contribuez

5. creez une branche pour développer le code correspondant
6. publiez cette branche sur votre dépôt
7. Soumettez un Pull Request ou PR (Github) Soumettez un Merge Request ou MR (Gitlab)
8. Attendez le verdict. Si refus, retour à 5 ...

motifs de refus

- Merge impossible (rebasage nécessaire …)
- Style non conforme, manque de commentaires
- Manque de tests ou tests qui échouent

integration continue

- certains tests ou programmes peuvent être automatiquement lancés quand certains fichiers sont mis à jour sur le dépôt
- spécifié dans un fichier .gitlab-ci.yml sur Gitlab
- similaire à un Dockerfile

TDD - méthodes agiles

- Test-Driven Development
- Kent Beck 2003
- premiers concepts d'extreme programming
- méthodes agiles

Agile

- Les méthodes agiles constituent de nouveaux modes d'organisation
- Pas seulement en informatique
- Industrie automobile, logistique, etc.

Méthodes les plus connues

- Pair Programming
- Test Driven Development
- Scrum
  - Scrum Master
  - Product Owner
  - Sprints
  - StandUp meetings
  - Retrospectives

Standup Meetings

But de ces meetings

- Fournir un feedback (retrospective)
- Organiser les étapes suivantes
- Etre dans une position “active”

Backlogs (Todo lists)

Overloaded backlog

Figure 8: backlog un peu chargé

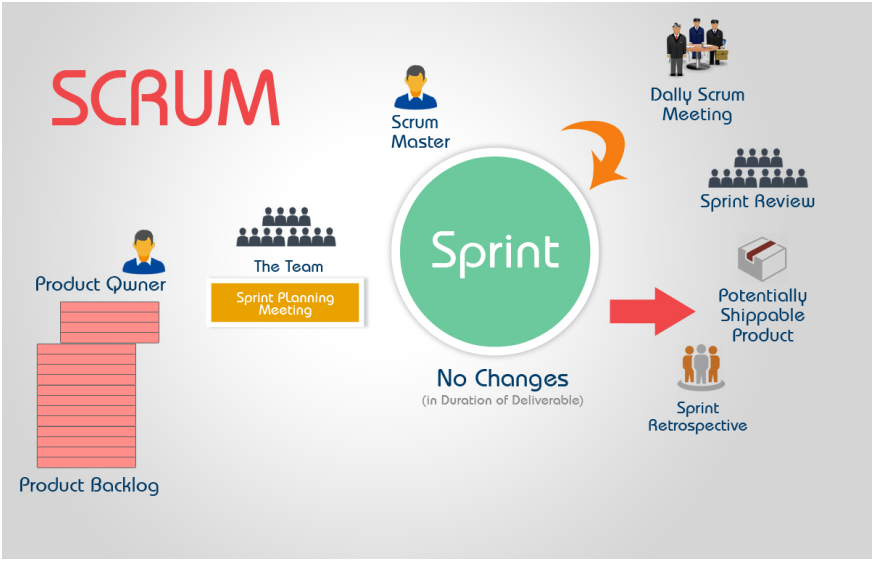


Figure 9: Eléments Scrum

Scrum

- Pierre angulaire :
  - items de Backlog
  - Tests
  - TDD et versionnage
- Groupes :
  - Typiquement 4-5 personnes
  - certains projets concernent des centaines de personnes (noyau linux, OS, etc.)

Avant tout: les Tests

- Différentes sortes de tests
- exemples
- Pourquoi le TDD ?

Différentes sortes de tests

```
En informatique, un test est une procédure de vérification partielle d'un syst
```

Il y a une grande diversité de tests, concentrons nous sur les 3 principales catégories

Les 3 principaux types de tests

- tests unitaires : s'appliquent à une méthode ou fonction isolée du reste du système (d'où le nom d'unitaire)
- tests d'intégration : comme les tests unitaires, mais sans êtres séparés du reste du système.
- tests fonctionnels : On teste une fonctionnalité complète en décrivant une sucession d'actions effectuée par un utilisateur de l'application.

exemples

- Pour tester une fonction, on fournit une liste d'entrées/sorties
- Pour tester une fonctionalité, on peut décrire son comportement attendu en écrivant une User Story comme: "Alice se rend sur la page d'accueil du site, voit 2 zones de saisie de nombres qu'elle complète, puis clique sur le bouton ajouter, et constate avec plaisir que le résultat de l'addition des 2 nombres s'affiche dans une zone encadrée de vert au milieu de la page"
- Voir TDD with Python

Test-Driven Development

- si on formalise l'idée de penser d'abord en termes d'exemples d'entrées/sorties examples avant de coder
- on arrive naturellement au TDD :
  - definir les tests
  - puis on code

Automatiser les tests

Tous les langages ont leur librairie de tests unitaires

- phpUnit en PHP
- JUnit en Java
- pytest ou unnitest en Python
- GoogleTest ou Boost en C++

2021 Gérard Rozsavolgyi gerard.rozsa @ gmail.com