# Informatique pour l'entreprise

## Gestionnaire de versions

Marie Pelleau & Olivier Baldellon
marie.pelleau@univ-cotedazur.fr,
olivier.baldellon@univ-cotedazur.fr

30 janvier 2023

# Développement collaboratif

## Situation

- On travaille à plusieurs ($\geqslant 2$), en parallèle.
- On ne se rencontre pas forcément.
- On doit montrer à tout moment l'avancement.
- On doit parfois revenir en arrière.
- On ne doit rien perdre.
- On doit pouvoir savoir qui a fait quoi.

- Solution manuelle : **hors de question !**
- Utiliser un **système de gestions de versions.**

# Observer les différences

## Principes

- Basé sur les outils de comparaisons de fichiers.
- Mémorisation des parties qui ont changé :
  - supprimer,
  - ajouter,
  - modifier = supprimer + ajouter,
  - les parties modifiées sont regroupées en paquets.

## En pratique

- Le système de gestions de versions va « cacher » le fonctionnement.
- **Choix des fichiers à stocker / indexer**.

# En cas de conflit(s)

## Fusion automatique

si les changements portent sur des lignes différentes.

## Fusion par les contributeurs

dans les autres cas.

- Besoin de comprendre.
- Besoin de communiquer.

# Gestion centralisée (par exemple : svn)

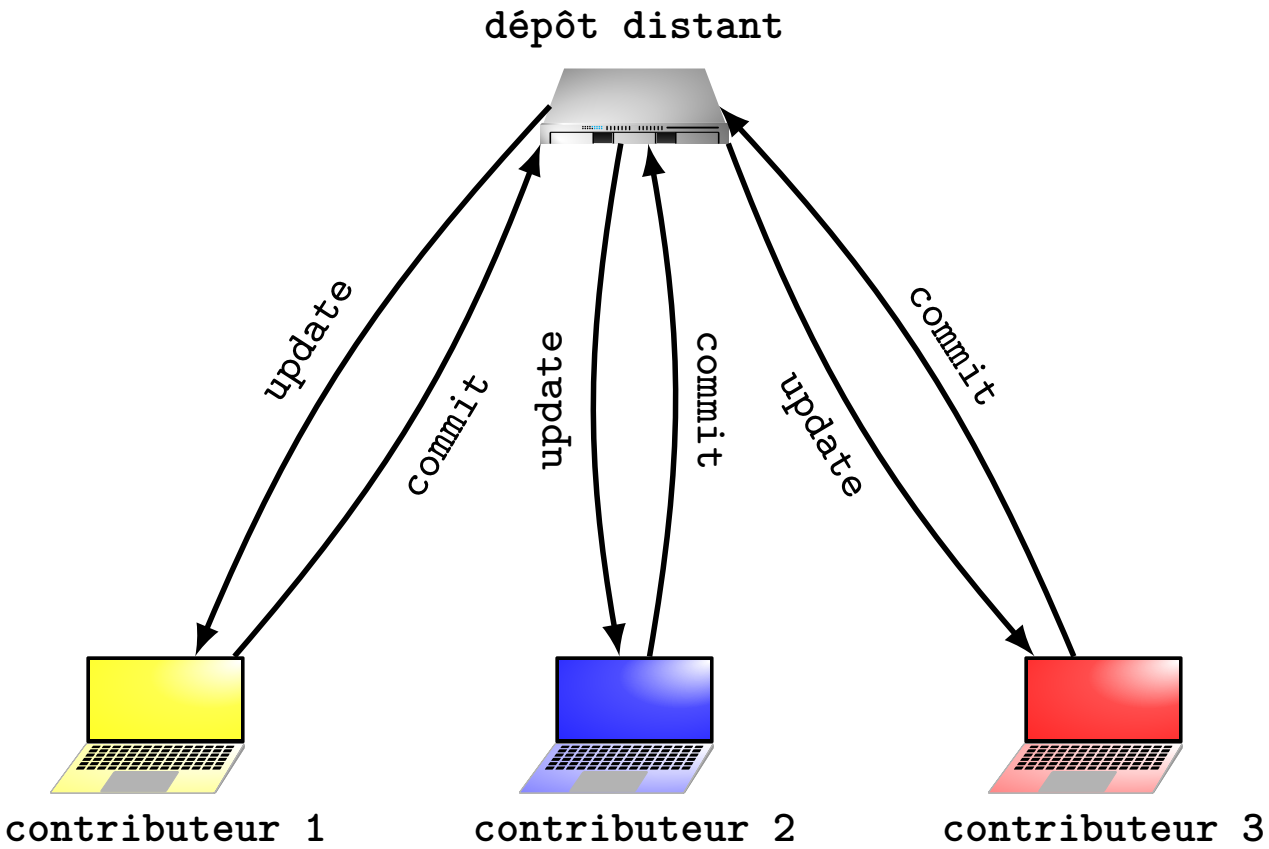



Figure – Fonctionnement centralisé, par exemple subversion (`svn`)

# Gestion décentralisée (par exemple : git)

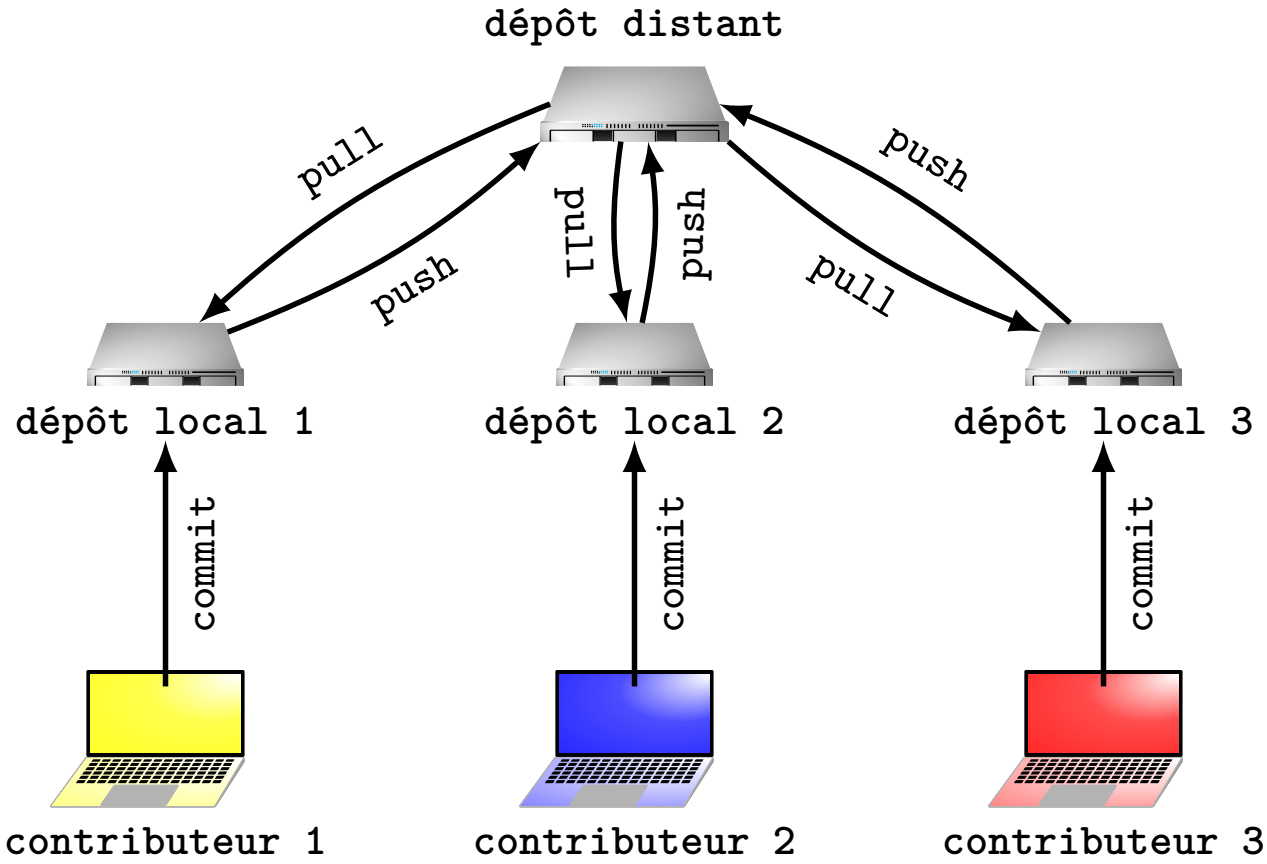



Figure – Fonctionnement décentralisé, par exemple (`git`)

**C'est le système avec lequel nous allons travailler.**

# Mise à jour avant de propager

- Mes modifications **vs** leurs modifications.
- **Conflits** : modifications des mêmes lignes.
- Si j'envoie sans vérifier : risque de conflits non résolus dans le dépôt distant $\longrightarrow$ problème pour tout le monde.

## Vérifications avant d'envoyer

1. Je récupère les modifications des autres.
2. Je résous les conflits.
3. Je peux envoyer.

# Dans un dépôt, on trouve ...

- le code source (`.c`, `.h`, `.py`),
- les ressources (fichiers images, données, ...),
- les fichiers techniques pour la gestion des dépendances et la compilation (par exemple, **Makefile**),
- la documentation (par exemple, fichiers **markdown**),
- le fichier `.gitignore`.

# Dans un dépôt, on **ne** trouve **pas** ...

- des fichiers générés (`.pyc`, `.o`, ...),
- des fichiers de configurations propres à une machine (de votre éditeur par exemple),
- des fichiers trop gros,
- tout fichier extérieur au projet.

## Le fichier `.gitignore`

- Il est **présent** dans le dépôt.
- Il explique quels fichiers ne doivent pas être dans le dépôt.

# git : présentation

## Généralités

- Créé en 2005 par Linus Torvalds (le créateur de Linux).
- Dernière version : 2.39.1 (17 janvier 2023).
- Multiplateforme (disponible sous Linux, Mac, Windows).

## installation

- Sous Linux, `sudo apt install git-all`
- Sous Windows, `http://git-scm.com/download/win`
- Sous Mac, essayez de lancer `git --version`
  Il faudra peut-être installer `Xcode Command Line Tools` au préalable.

## Commande ou interface graphique ?

Comprendre les commandes : bien pour débuter. Après, à vous de voir.

# git : qui vous êtes

## Fichiers de configuration

- globale : `~/.gitconfig`
- locale : `.git/config`

## Configuration (globale)

- `git config --global user.name "Votre nom"`
- `git config --global user.email "votre.adresse"`
- `git config --global color.ui auto`
  Active la colorisation de la sortie en ligne de commande.
- `git config --global core.editor "[éditeur]"`
  Dire quel `[éditeur]` utiliser, exemples :
  - `code --wait` (VScode)
  - `emacs`

# Création ou récupération

## Créer ou récupérer un dépôt

- Initialisation : `git init` dans le répertoire où vous voulez créer votre dépôt.
- Récupération : `git clone [chemin_vers_le_dépôt]`

## Exemple

```
git clone git@github.com:mpelleau/elements.git
```

# Opérations de base : git fetch et git pull

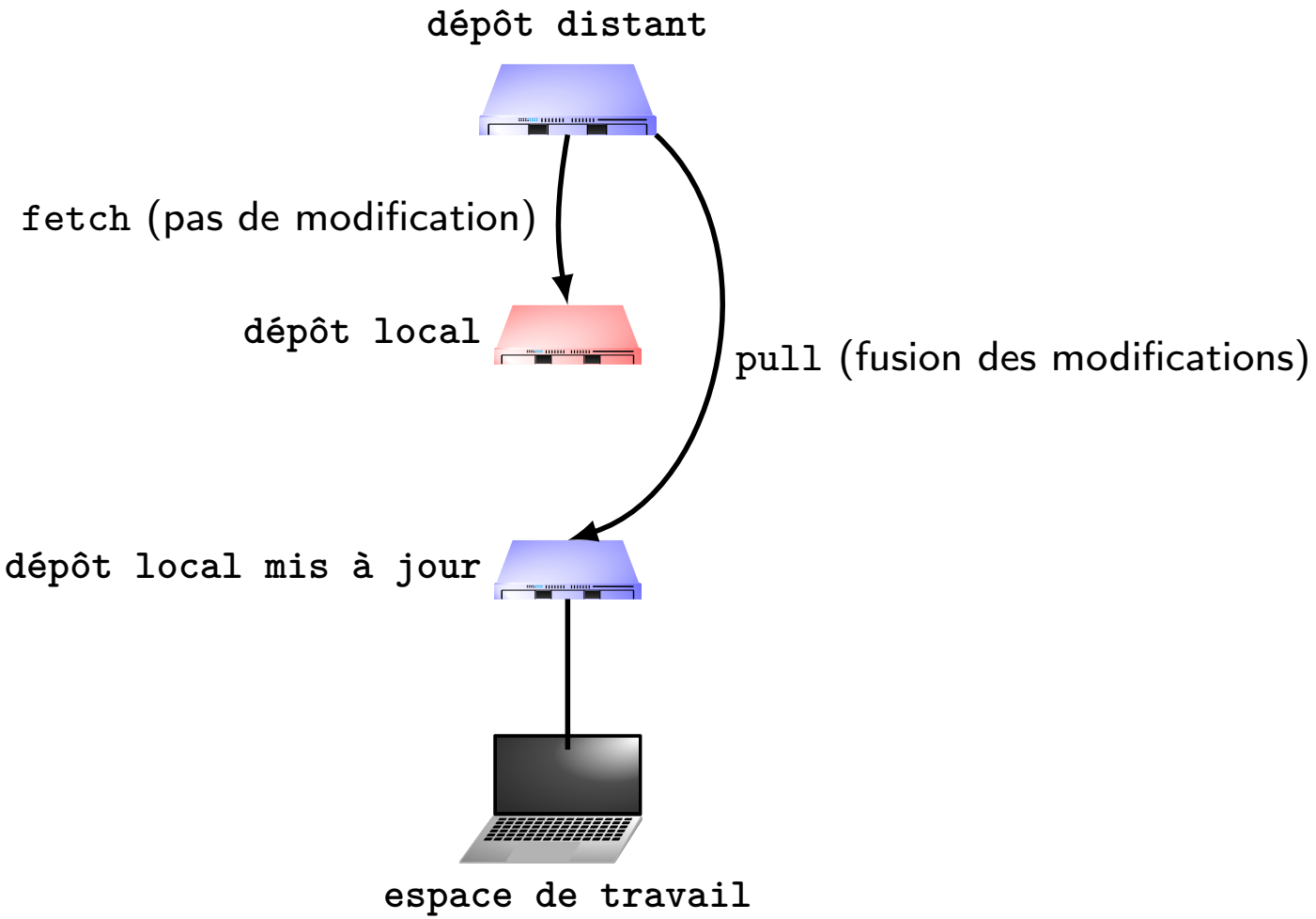



Figure – `git fetch` et `git pull` pour mettre à jour le dépôt local

# Opérations de base : produire

- `git add` : ajouter un fichier sur lequel on travaille pour qu'il soit indexé.
  - `git add fichier1 fichier2 fichier3`
  - `git add -A` : ajouter tous les fichiers
- Variantes :
  - `git rm fichier1` : supprimer le `fichier1`
  - `git mv fichier1 nouveau_nom_ou_chemin` : déplacer / renommer
- `git commit` : toutes les modifications sont enregistrées
  - `git commit fichier1 -m "Commentaire qui explique ce commit."`
    Les modifications sur `fichier1` sont enregistrées.
  - `git commit -a -m "Commentaire qui explique ce commit."`
    Les modifications sur **tous les fichiers indexés** sont enregistrées.
- `git push` : envoyer sur le dépôt distant.
  On exécute `git pull` **auparavant**.

# Fichiers indexés et leurs états

## Et les répertoires ?

- On ne s'occupe **que des fichiers** avec `git`.
- Seuls les répertoires qui contiennent (au moins) un fichier indexé apparaissent.

## États d'un fichier

- **Modifié** : le fichier a été modifié, mais ces modifications n'ont pas été `commit`.
- **Staged** : le fichier a été ajouté (`add`).
- **Committed** : le fichier est stocké dans le dépôt local.

# Informations sur l'état

## Quelques commandes de base de comparaisons

- `git status` : informations sur l'état des fichiers.
- `git reflog` : historique des commits (court).
- `git log` : historique des commits (détaillés).
- `git diff` : différence entre l'espace de travail et la copie locale du dépôt distant.

# tag

## Principe

- Pour ajouter une étiquette sur un commit.
- Pour indiquer une version particulière.

## En pratique

- `git tag nomDuTag` : pour créer le tag. Reste local pour l'instant.
- `git push origin nomDuTag` : pour transmettre le tag sur le dépôt distant.

# Gestion des conflits : synopsis

## Gérer les conflits : une façon de procéder.

- Je fais des modifications.
- J'exécute `git stash` pour mettre (temporairement) mes modifications de côté.
- J'exécute `git pull`
- J'exécute `git stash pop` pour reprendre mes modifications et les appliquer.
- En cas de conflit(s), je le(s) résous.
- J'indexe (`add` / `commit`) ce qui a été résolu.
- Je peux `push`

# Jouer avec l'historique

## Tant qu'on n'a pas `push`

- Changer le dernier `commit` : `git commit --amend`
  quand j'ai oublié un commentaire, fait une faute...
- Annuler le dernier `push` (qui n'a pas été « pushed ») :
  - `git reset HEAD^ --soft` : annule l'action `git commit`,
  - `git reset HEAD^ --mixed` : annule l'action `git commit` et `git add`,
  - `git reset HEAD^ --hard` : annule l'action `git commit`, `git add` et les modifications (perte).
- On peut changer l'ordre des commits, en effacer. Exemple :
  `git rebase -i HEAD~3`

## Forcer

Si on a exécuté `git push`, il faudra « forcer le `push` », ce qui est possible, mais **déconseillé**.

# Se positionner dans l'arbre

Exemple d'arbre des commits : `git log --graph --decorate --oneline --all`

```
*   74de497 (HEAD -> main, origin/main, origin/HEAD) Fusion des modifications
|\
| * 34755e5 Explication des limites.
* | 9763c01 Explications python.
* | d2bd8d6 De la couleur.
|/
* e9a90fd Ajout explications. Fixes #7
* bd9c188 (tag: v0.2) Affichage avec des cases de tableau. Fixes #6
* 732de1a Correction coquille. Closes #5
* 481e300 Traduction française. Closes #4
* baef50a (tag: v0.1) Une classe pour les éléments. Fixes #3
* 67da7fb Début fonction classifier. Premier affichage pour #2.
* ec083f2 Ajout des éléments jusqu'à 20. Fixes #1
* 2d169a2 Mise à jour du README.
* 5747141 Initial commit
```

- `git checkout` : se positionner dans l'arbre
- Habituellement `HEAD` pointe sur le dernier commit.
- `git checkout` permet de se détacher de `HEAD`.
- Les actions seront faites à partir de là où est `HEAD`.

# Navigation dans l'arbre

## Que mettre après git checkout

- À partir d'un point de départ, on peut remonter au parent avec ^ (exemple : `HEAD^`).
- À partir d'un point de départ, on peut remonter avec `~nb`
  - Exemple : `HEAD~3` désigne le parent du parent du parent de `HEAD`.
- Ou alors on utilise le SHA-1.

# github

## Présentation

- `github.com` est un service d'hébergement et de gestion de versions.
- Il permet d'héberger un projet `git` et ajoute des fonctionnalités (et une interface).
- Ce n'est pas le seul système. Autres exemples : gitlab, framagit, bitbucket, ...

## Créer un compte

- Vous pouvez créer un compte sur github, c'est gratuit et cela servira pour les TP.
- Ajoutez votre clé publique `ssh` (menu Settings, SSH and GPG keys) pour travailler avec git de façon sécurisée.

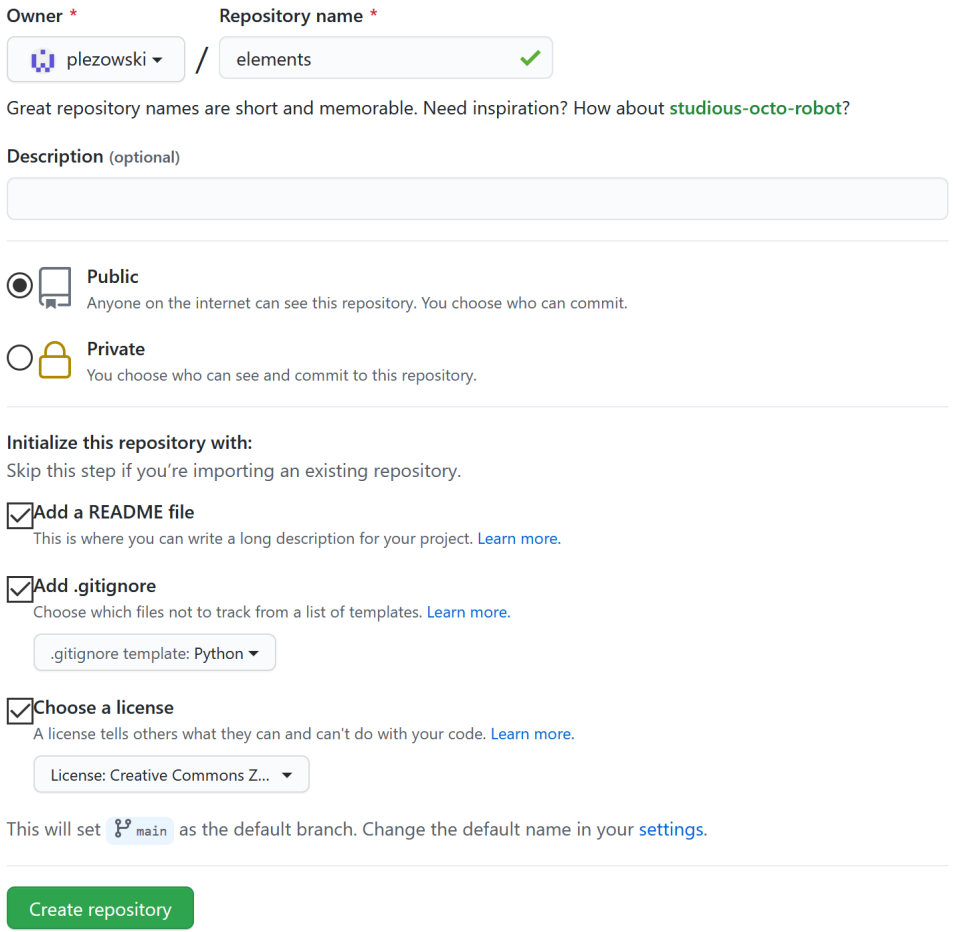
# Créer un dépôt github

## Créer un dépôt sur le site

- qui génère automatiquement l'arborescence `git`,
- qui peut créer un fichier `.gitignore`,
- qui peut créer un fichier de documentation `README.md`.

## Cloner sur votre machine

- `git clone https://github.com/plezowski/elements.git`
- ou `git clone git@github.com:plezowski/elements.git` (avec clé `ssh`, mieux)



# Les issues

## Issues

- Les issues permettent de suivre les tâches à accomplir, bugs, améliorations du projet.
- Disponibles sur l'onglet « Issues » de github.
- Chaque issue a un #numéro.
- On peut **assigner** une (ou plusieurs) personne(s) à une issue.

## Labels

- Les labels (étiquettes) servent à organiser les issues.
- On peut créer d'autres labels.

## Milestones

- Les issues peuvent être regroupées en milestones.
- Les milestones peuvent correspondre à une version du projet.

# Liaison entre commit et issue

## Organiser vos commits

- Format standard d'un commit : explication du commit et #numIssue.
- Pour fermer l'issue en même temps que `push`, préciser `fixes #numIssue` (ou autre verbe comme closes).

## Vous organiser

- Créer les issues **avant** d'écrire le code.
- Essayer de réfléchir autant que possible à « long terme ».
- Vous n'êtes pas limité(e) sur les commits, faites des commits fréquents, avec des descriptions précises.

# Quelques outils

## tags

- Dans github, les tags produisent des **releases**.
- Cela donne un accès direct pour télécharger le code à cette version.

## Comparaisons

- Permet de voir les effets d'un commit.
- Exemple : `https://github.com/plezowski/elements/commit/baef50af524f9bd42cef2cca4dc9b75a1e08249c`

## Statistiques

- Voir onglet « Insights ».

# Les fichiers markdown

## Présentation

- Markdown est un langage de balisage léger.
- github affiche automatiquement les fichiers `.md` avec une « belle présentation ».
- Exemples : `README.md` créé automatiquement, plus généralement, les descriptions des issues.

## Quelques exemples

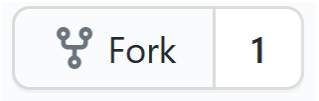
- Mettre en *italique* : `*italique*` ou `_italique_`
- Mettre en **gras** : `**gras**` ou `__gras__`
- Écrire du `code` : `` `code` ``
- Écrire un titre : `#Titre important`, `## Sous-titre`, ...
- Et bien davantage (par exemple : listes, hyperliens, tableaux, ...)

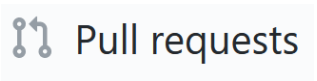
# pull-request

## Objectif

Suggérer des modifications à un projet public.

## Étapes

- Aller sur la page du projet et cliquer sur 

- Cloner le fork qui vient d'être créé.
- Faire les modifications sur le fork, `commit` et `push`.
- Sur l'interface du projet d'origine, sélectionner 

, puis « compare across forks », les dépôts, puis « Create pull request ».
- Les modifications peuvent alors être validées (ou non) par les dépositaires du projet d'origine (« merge pull request »).
- Le fork peut alors être supprimé.

# Introduction

- Pour gérer les projets avec github, nous avons utilisé l'interface web de github (autrement dit le site internet `github.com`)
- Gérer le projet :
  - Créer le dépôt
  - Créer des issues
  - Ajouter une collaboratrice ou un collaborateur
- On peut faire tout ça en ligne de commandes

# curl

- Nous avons vu la commande `wget` pour télécharger un fichier depuis le terminal
- La commande `curl` est une interface en ligne de commande

```
curl https://raw.githubusercontent.com/mpelleau/corps-celestes/main/mercure.md
```

```
# Mercure

Mercure est la planète la plus proche du soleil.

![Icone de mercure](mercure.png)

Icone créée par [monkik](https://www.flaticon.com/authors/monkik) trouvée sur [flaticon](https
      ://www.flaticon.com/).

## Caractéristiques

- Masse : 3.301 x 10<sup>23</sup> kg
- Diamètre : 4879.4 km
- Distance au soleil :
  - Aphélie : 7 x 10<sup>7</sup> km
  - Périhélie : 4.6 x 10<sup>7</sup> km
  - Demi-grand axe : 5.8 x 10<sup>7</sup> km
- Symbole : &#x263F;
```

- Récupère le fichier `mercure.md` dans le terminal

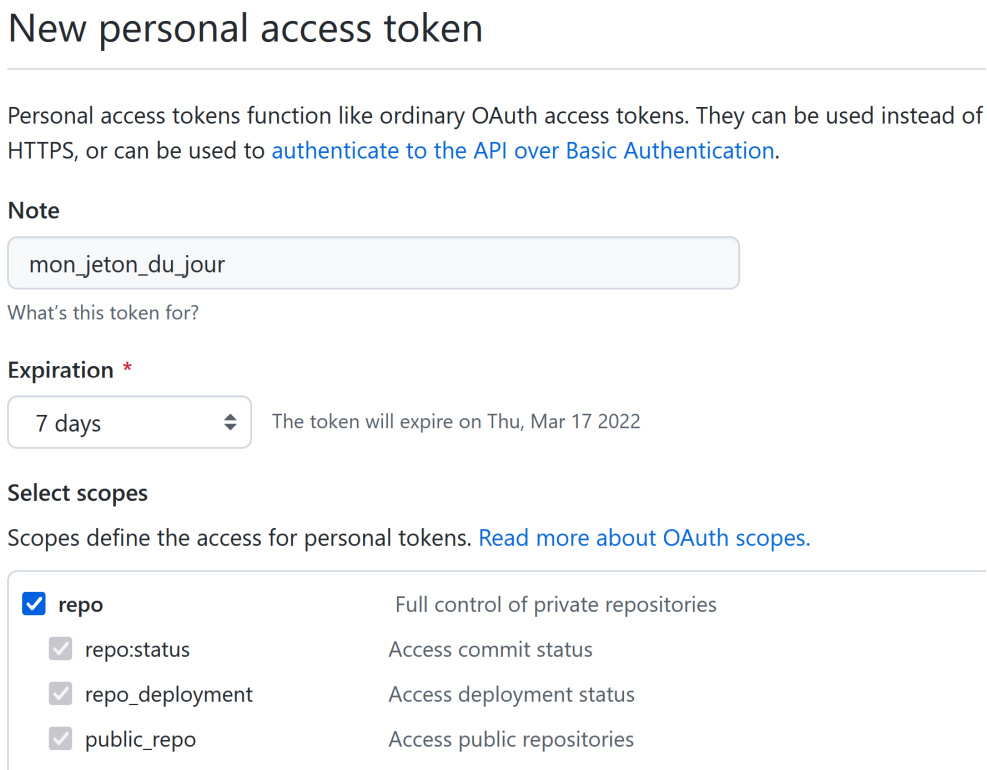
# Authentification

Plusieurs techniques pour obtenir des autorisations.

- Mot de passe
- Clés ssh
  - une clé privée (qu'on garde pour soi)
  - une clé publique (qu'on peut diffuser)
- OAuth n'est pas vraiment un protocole d'authentification
  - c'est un protocole de délégation d'autorisation
  - permet de donner à un site ou un logiciel certaines de ses autorisations
  - l'autorisation est donnée sous forme de jeton (token)

# Jetons (tokens) de github

- Sur la page https://github.com/settings/tokens, on peut créer un nouveau jeton
- On peut contrôler les permissions
- On doit copier le jeton sur l'instant, il ne sera plus visible ensuite



# L'API REST de github : grands principes

- API : interface de programmation d'application
- Interface pour se connecter au service d'un logiciel ou d'un site (ici `github.com`) pour programmer ou échanger des fonctionnalités
- On va communiquer avec `github.com` par l'intermédiaire de la ligne de commandes, plutôt que par un navigateur internet
- Les retours vont aussi être dans le terminal, mais les effets vont bien être sur nos dépôts (donc visibles aussi dans le navigateur internet)

# Premiers échanges

- Premier contact :
  `curl -X GET https://api.github.com/zen`

```
Mind your words, they are important.
```

- Maintenant, récupérons les informations sur l'utilisatrice `mpelleau` :
  `curl -X GET https://api.github.com/users/mpelleau`

```
{
  "login": "mpelleau",
  "id": 4224775,
  ...
  "type": "User",
  "site_admin": false,
  "name": "Marie Pelleau",
  "company": "Université Côte d'Azur",
  "blog": "http://www.i3s.unice.fr/~mpelleau/",
  ...
}
```

# Plus de détails : les en-têtes et jeton

- Ajoutons l'option `-i` pour avoir les en-têtes :
  `curl -i -X GET https://api.github.com/users/mpelleau`

```
HTTP/2 200
server: GitHub.com
date: Thu, 10 Mar 2022 18:40:42 GMT
content-type: application/json; charset=utf-8
...
x-ratelimit-limit: 60
x-ratelimit-remaining: 57
...
```

- Les requêtes sont limitées en nombre (60 par jour), mais on peut avoir une plus grande limite (5000) si on se connecte.

```
jeton=ghp_1c3iW2s2raBz88Fos5APD6QYOOvkh349XAne
curl -u moncompte:$jeton -i -X GET https://api.github.com/users/mpelleau
```

```
...
x-ratelimit-limit: 5000
x-ratelimit-remaining: 4999
...
```

- Le jeton (token) est stocké dans la variable `jeton`

# Créer un dépôt

- `curl -X POST -u moncompte:$jeton https://api.github.com/user/repos -d '{"name":"superdepot", "private":true}'`

```
{
  "id": 468473102,
  "node_id": "R_kgDOG-xVDg",
  "name": "superdepot",
  "full_name": "moncompte/superdepot",
  "private": true,
  ...
}
```

- On utilise `POST` (et pas `GET`) dans ce cas

# Les issues

- Voir les issues

```
curl -X GET -u moncompte:$jeton https://api.github.com/repos/plezowski/elements/issues
```

```
{
  "url": "https://api.github.com/repos/plezowski/elements/issues
      /9",
  ...
  "number": 9,
  "title": "Ajouter les éléments jusqu'à 30 de la classification
      ",
  ...
}
```

- Créer une issue

```
curl -X POST -u moncompte:$jeton https://api.github.com/repos/moncompte/superdepot/issues -d '{"

title":"Faire quelque chose", "body":"il faudrait faire quelque chose", "labels": ["travail"]}'
```

```
{
  "url": "https://api.github.com/repos/moncompte/superdepot/
      issues/1",
  "repository_url": "https://api.github.com/repos/moncompte/
      superdepot",
  ...
}
```

# Collaboration

- Ajouter une collaboratrice

```
curl -X PUT -u moncompte:$jeton https://api.github.com/repos/moncompte/superdepot/collaborators/
mpelleau
```

- Bien sûr, la collaboratrice doit accepter l'invitation
- Supprimer une collaboratrice

```
curl -X DELETE -u moncompte:$jeton https://api.github.com/repos/moncompte/superdepot/collaborators/
mpelleau
```

# Pour aller plus loin

- Voir la documentation en ligne https://docs.github.com/en/rest
- Exemple : supprimer un dépôt
  ```
  curl -X DELETE -u moncompte:$jeton https://api.github.com/repos/moncompte/superdepot
  ```
- NB : suppression sans confirmation, pour les utilisatrices et utilisateurs avertis !

# API REST

- REST : REpresentational State Transfer
- Architecture logicielle définissant un ensemble de contraintes pour des services web
- Utilise souvent les méthodes GET, POST, PUT, DELETE (et autres méthodes http non vues ici)
- Contraintes :
  - Séparation client-serveur
  - Sans conservation de l'état
  - Avec mise en cache
  - En couche

# Bilan

## Documentation / aide-mémoire

- Le site de documentation `https://git-scm.com/docs/`
- Aide-mémoire de github `https://training.github.com/`
- Un autre aide-mémoire `https://ndpsoftware.com/git-cheatsheet.html`
- Aide-mémoire sur markdown pour github : `https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet`

## Seulement un aperçu...

`git` a bien d'autres fonctionnalités, par exemple

- les **branches** pour développer à part de nouvelles fonctionnalités,
- `git blame` pour voir qui a écrit quel morceau de code,

mais pour un premier contact avec `git`, nous nous concentrerons sur les fonctions de base.