



UMR botanique et Modélisation de l'Architecture des Plantes et des végétations

Introduction à GIT

Atelier MAIA P3M 04 avril 2019

Nicolas B., François G., Philippe V., Rémi V., Ghislain V.



cirad



INRA
SCIENCE & IMPACT



Institut de recherche
pour le développement



UNIVERSITÉ
DE MONTPELLIER



Programme de la matinée

- **09h00-09h45** : Présentation introductive
- **09h45-10h00** : Pause
- **10h00-11h00** : Exercice 1, travail sur dépôt GitHub personnel, commandes de base en mode “invite de commande”
- **11h00-11h15** : Démonstration exercice 1 avec interface graphique
- **11h15-12h00** : Au choix de chacun
 - Reprise Exercice 1 en mode “interface graphique”
 - Exercice 2, gestion d’un conflit
 - Exercice 3, travail en binôme sur dépôt GitHub partagé



Présentation introductive

1. Nécessité et utilité de Git
2. Historique du versionnement et des outils
3. GIT notions de base
4. GIT commandes de base
5. Ecosystème GIT

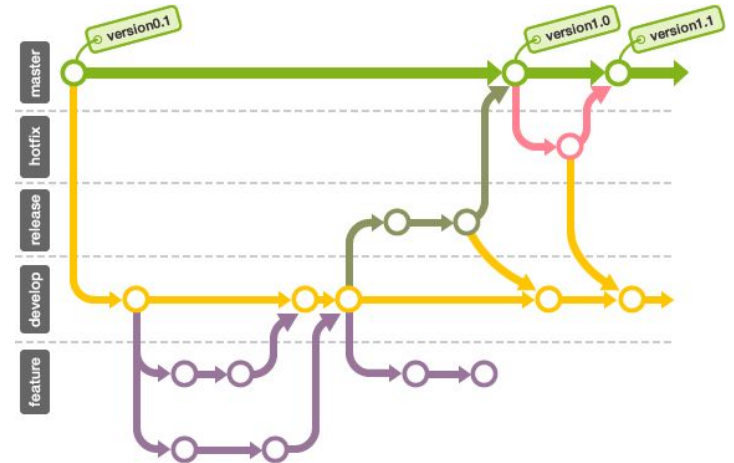
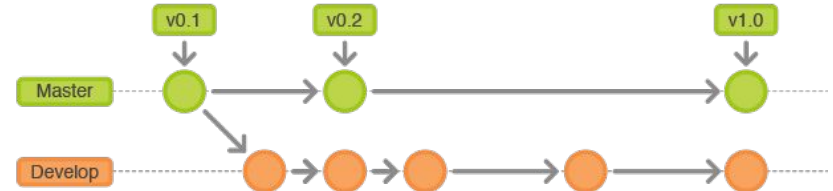


Nécessité et utilité de Git

Nécessité et utilité de Git

- **Usage personnel - utilité première**

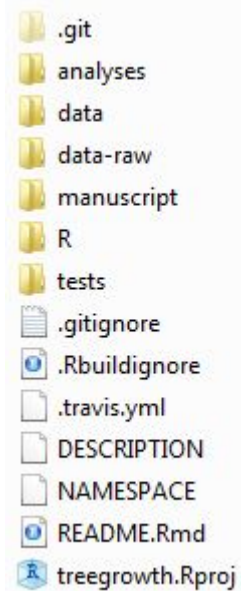
- Versionner son travail
 - Enregistrement des modifications successives (“commit”)
 - Possibilités de revenir à des modifications antérieures
 - Versionnement (v0.1)
 - Branche maître, branche de développement



Nécessité et utilité de Git

- **Usage personnel - autres avantages**

- Organiser son travail sous forme de répertoire:
 - Un projet (logiciel, article) = un répertoire Git
 - Organisation du répertoire (data/scripts/outputs)
- Sauvegarde sur serveur du travail (ex. GitHub), accès distant
- Synchronisation multipostes (bureau, domicile)
- Partage avec la communauté scientifique: codes, données (ou script de téléchargement des données)
- Reproductibilité des résultats de recherche



⇒ **Bonnes pratiques (qualité/gain de temps)**

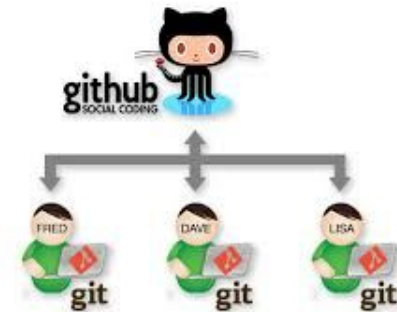
Source:

<https://github.com/Pakillo/template>

Nécessité et utilité de Git (et du versionnement)

● Usage collaboratif

- Permet de faire tout ce qui a été dit précédemment mais de façon collaborative
- Modification d'un répertoire Git à plusieurs
- Récupération des modifications faites par d'autres utilisateurs
- Gestion des conflits
- Ex: noyau Linux, GRASS GIS, GSL GNU, etc.





Historique du versionnement et des outils

Historique du versionnement et des outils

La **gestion de versions** (*version control* ou *revision control*) :

- consiste à maintenir l'ensemble des versions d'un ou plusieurs fichiers ;
- concerne surtout la gestion des codes sources (mais pas que : documents texte, binaires, ...).

Outil : **logiciel de gestion de versions** (VCS : *Version Control System*) :

- stocker un ensemble de fichiers en conservant l'**historique de leurs modifications** : **traçabilité**, **retour** à une version antérieure ;
- travail **collaboratif** : possibilité pour un utilisateur de récupérer / partager des modifications.

Vocabulaire:

- **modification** : évolution entre deux **versions (révisions)** d'un fichier ;
- **dépôt** (distant) : espace de stockage géré par le logiciel de gestion de version contenant les fichiers versionnés (archivés) ;
- **copie locale** : espace local contenant une copie des fichiers du dépôt distant.

Historique du versionnement et des outils



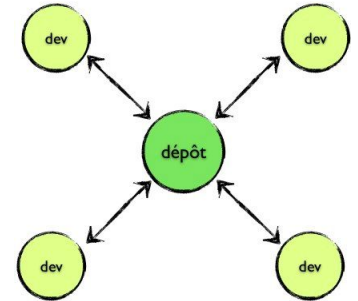
Outils de première génération : gestion de versions locale

- **SCCS** (Source Code Control System) : *première version : 1972, dernière version : 2019, licence propriétaire puis licence libre*
 - système de commandes ;
 - permet à plusieurs utilisateurs de **suivre les modifications** d'un code source, fichier texte, binaire,... ;
 - enregistre toutes les modifications d'un fichier dans un fichier "historique" : permet de récupérer toute version d'un fichier ;
 - ne fonctionne que sur des **fichiers individuels** (ne supporte pas des modifications affectant de multiples fichiers) ;
 - n'autorise qu'**un utilisateur à la fois** à modifier un fichier (verrou) ;
 - intégré à la plupart des versions UNIX.
- **GNU RCS** (Revision Control System) : *première version : 1982, dernière version : 2015, licence libre*
 - alternative libre à SCCS : ne fonctionne que sur des **fichiers individuels**, n'autorise qu'**un utilisateur à la fois** à modifier un fichier → **pas adapté à de gros projets** ;
 - par rapport à SCCS : interface utilisateur plus conviviale, récupération plus rapide des versions.
- **Problème de sécurité** : les utilisateurs de SCCS / RCS peuvent éditer les fichiers de contrôle de version → client-serveur

Historique du versionnement et des outils

Outils de seconde génération : gestion de versions centralisée (CVCS : Centralized Version Control System)

- Caractéristiques : architecture **client-serveur** :
 - un **serveur** stocke la version actuelle d'un projet et son historique : un seul **dépôt centralisé** des versions ;
 - les clients se connectent au serveur pour extraire une copie complète du projet (**copie locale**), travaillent sur cette copie puis archivent leurs modifications.
- Avantages : **simplicité** d'utilisation (un seul dépôt).
- Inconvénients : impossibilité d'effectuer une action (archivage de modifications, comparaison de versions de fichiers, ...) si le dépôt est inaccessible ou sans connexion au réseau.



Exemples :

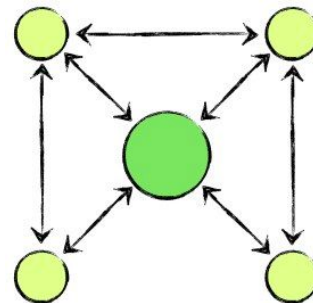
- **CVS** (Concurrent Versions System) : *première version : 1990, dernière version : 2008, licence libre*
 - successeur de RCS : **plusieurs personnes** peuvent travailler sur le même fichier **simultanément** (chacune dans sa copie locale) → **adapté à de gros projets** par rapport à RCS.
- **Subversion** : *première version : 2000, dernière version : 2019, licence libre*
 - successeur de CVS : numéro de version global (pour l'ensemble du projet) et pas par fichier, possibilité de renommer ou déplacer un fichier, **simple à utiliser**.



Historique du versionnement et des outils

Outils de troisième génération : gestion de versions décentralisée (DVCS : Decentralized Version Control System)

- Caractéristiques :
 - chaque copie de travail est un dépôt complet qui contient tout l'historique ;
 - permet la communication entre les dépôts locaux.
- Avantages :
 - fonctionne en mode déconnecté ;
 - les opérations se font en local : plus rapide ;
 - plus de sécurité : il n'y a pas qu'un seul dépôt de référence.
- Inconvénients : plus difficile à utiliser que les CVCS (ex : Subversion) ?



Exemples :

- **BitKeeper** : première version : 2000, dernière version : 2017, licence libre ;
- **GNU Arch** : première version : 2001, dernière version : 2006 (non maintenu), licence libre ;
- **Darcs** : première version : 2002, dernière version : 2018, licence libre ;
- **Monotone** : première version : 2005, dernière version : 2014, licence libre ;
- **Git** : première version : 2005, dernière version : 2019, licence libre ;
- **Mercurial** : première version : 2005, dernière version : 2018, licence libre ;
- **Bazaar** : première version : 2005, dernière version : 2016, licence libre.





GIT notions de base

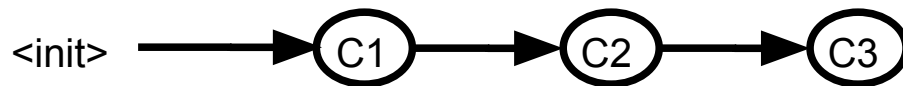


Dépôts

- C'est quoi ?
 - arborescence de fichiers versionnés
- Dépôt local
 - manipulé par commandes ou GUI (cwd : rép)
 - création (init) / télécharger une copie (clone)
- Dépôt distant
 - optionnel
 - lien (remote)

Versionner : séquence de commits

- Sauvegarde
 - état instantané des fichiers
 - date, auteur, description
 - parent/enfant, temps : historique



`git commit`



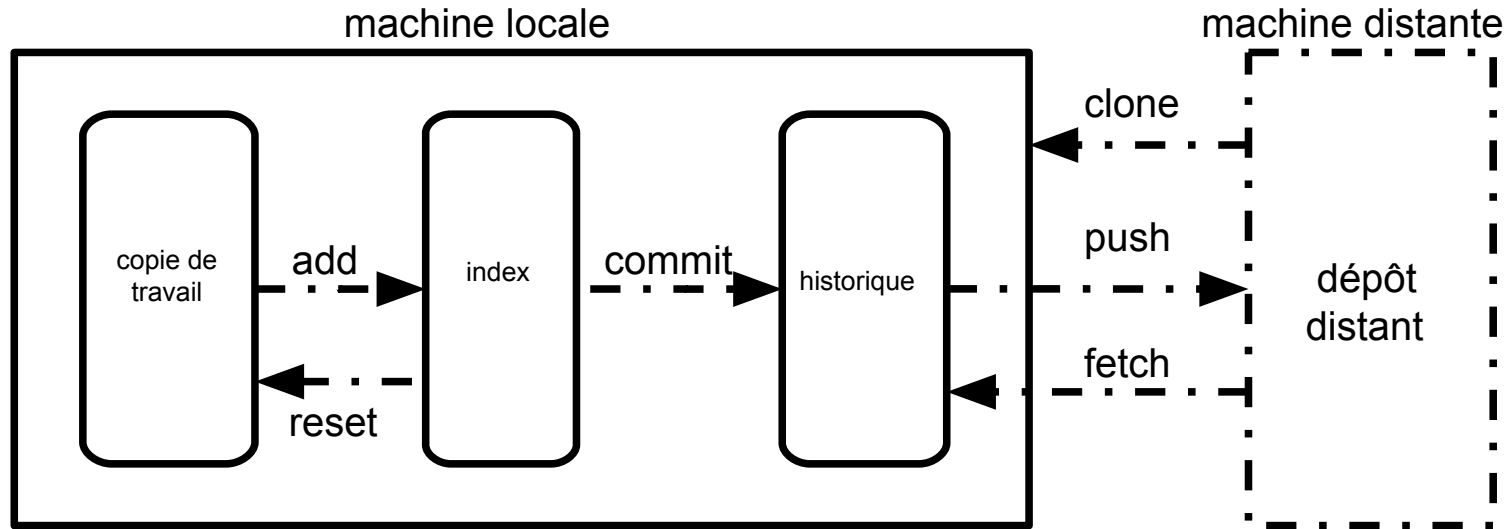
Étapes de travail

- Création du dépôt : init / clone
- Modification des fichiers
- Création de révisions : add / commit
- Partage : push / pull / fetch

Dépôt local

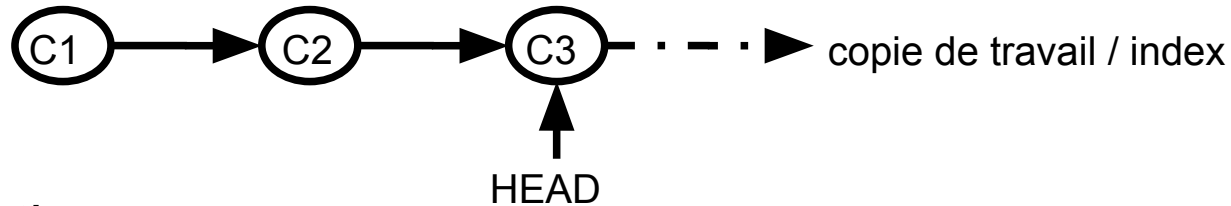
- Copie de travail
 - fichiers visibles, modifiés
- Index
 - préparation prochain commit
 - ajout, suppression
- Historique
 - commits
 - (lien dépôt distant)
 - .git

Local / distant

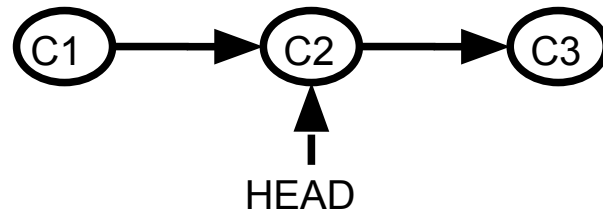


Navigation dans l'historique

- Pointeur HEAD



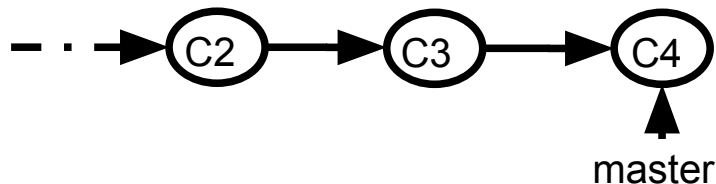
- Navigation



`git checkout <C2>`

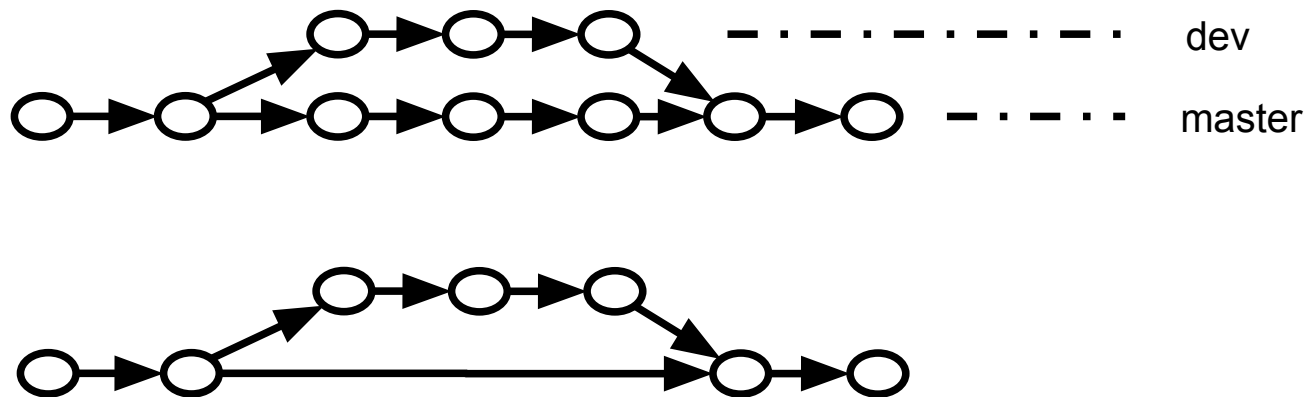
Branches

- Succession (arborescente) de commits
- Par défaut : master
- Création : `git branch <nom>`
- Les branches sont aussi des pointeurs



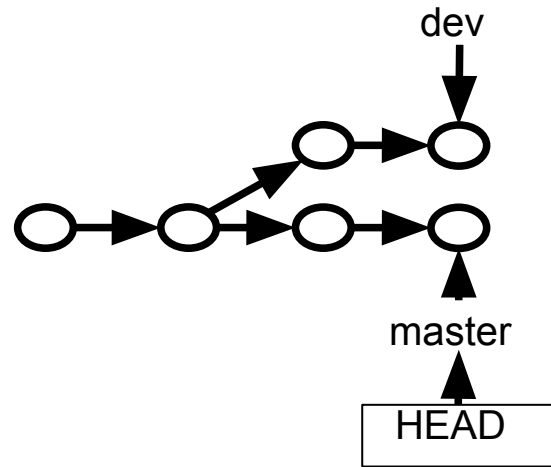


Branches

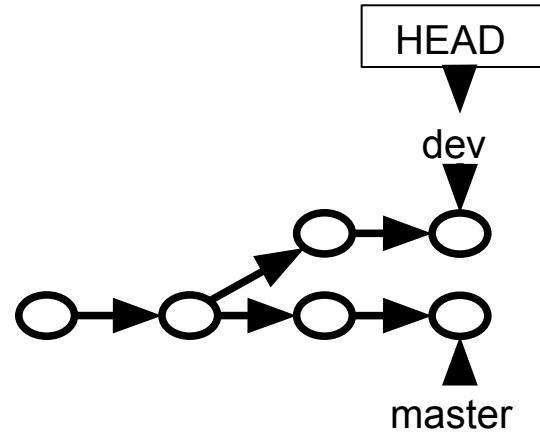


Branches

- Navigation entre branches



`git checkout dev`





GIT commandes de base



Commandes de base

- Création de dépôt
 - `git init [<répertoire>]`
 - `git clone <URL>`



Commandes de base

- Versionnement
 - `git add <chemin fichier>`
 - `git commit [-m message]`
 - `git reset <fichier> / git reset`
 - `!!\ git reset <commit>`
 - `git revert <commit>`



.gitignore

- fichier texte
- liste de fichiers que GIT doit ignorer
- au même niveau que .git/
- une entrée par ligne
- **ex :**

```
# commentaire pertinent
fichier1
repertoire2/
toto*
**/tata
```
- **git help ignore**



Etat du dépôt local

- git status
 - branche courante, état des fichiers
- git log
 - historique, position des pointeurs
- git diff
 - état des modifications
- git remote
 - liste/gestion des liens avec dépôts



Gestion des branches

- git branch
 - liste des branches locales (liens)
- git branch <branche>
 - création de branche
- git checkout <branche>
 - changer de branche



Trouver de l'aide

- git help : lister les commandes
- git help <commande> : aide de la commande
- Livre référence :
 - <https://git-scm.com/book/en/v2>
- Au sujet des branches :
 - <https://learngitbranching.js.org/>
- Tuto [OpenClassRooms Git/Github](#)

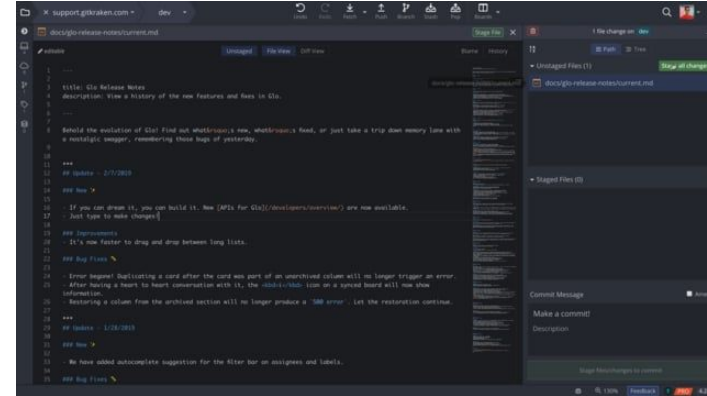
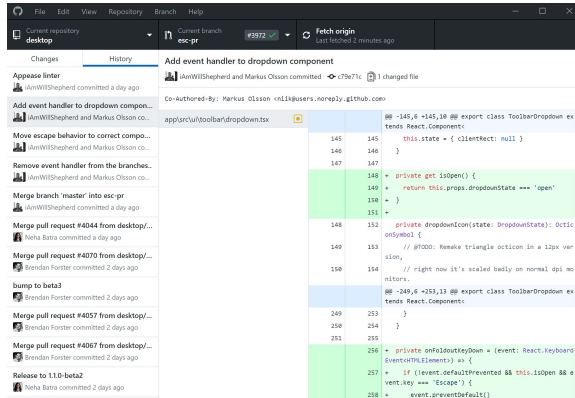


Écosystème GIT

Ecosystème GIT: outils

De nombreux outils autour de GIT:

- Interfaces utilisateurs (e.g. Github Desktop, GIT Kraken...)



- Intégration à des logiciels (e.g. RSTUDIO, ATOM...)
- Plateformes d'hébergement de dépôts en ligne (remote)

Ecosystème GIT: hébergement

Il existe de nombreux services d'hébergement en ligne pour les dépôts GIT:



GitHub



GitLab



ATLASSIAN

Bitbucket

Ces plateformes sont basées sur un modèle freemium (abonnement basique gratuit).
Attention, les dépôts peuvent être hébergés sur sol américain (zéro confidentialité).

Le MBB et l'IRD (DUNI) proposent déjà un hébergement utilisant le logiciel GITLAB.
Le CIRAD et RENATER vont en proposer un très vite.
La forge AMAPdev peut accueillir des dépôts GIT sur demande.

Ecosystème GIT: micro-services

Ces plateformes proposent d'autres micro-services tels que des services:

- d'intégration continue pour du test de code (e.g. CRAN checks pour packages R) et le déploiement d'applications:



Travis CI



AppVeyor



- de collaboration (issues, forking, pull request)
- statistiques sur les dépôts ou les collaborateurs
- reproductibilité de la recherche (e.g. DOI avec ZENODO)
- hébergement de sites web (e.g. documentation d'un package R)

La plupart de ces services sont gratuits pour un usage modéré.



Maintenant, au travail !





Programme de la matinée

- **09h00-09h45** : Présentation introductive
- **09h45-10h00** : Pause
- **10h00-11h00** : Exercice 1, travail sur dépôt GitHub personnel, commandes de base en mode “invite de commande”
- **11h00-11h15** : Démonstration exercice 1 avec interface graphique
- **11h15-12h00** : Au choix de chacun
 - Reprise Exercice 1 en mode “interface graphique”
 - Exercice 2, gestion d’un conflit
 - Exercice 3, travail en binôme sur dépôt GitHub partagé



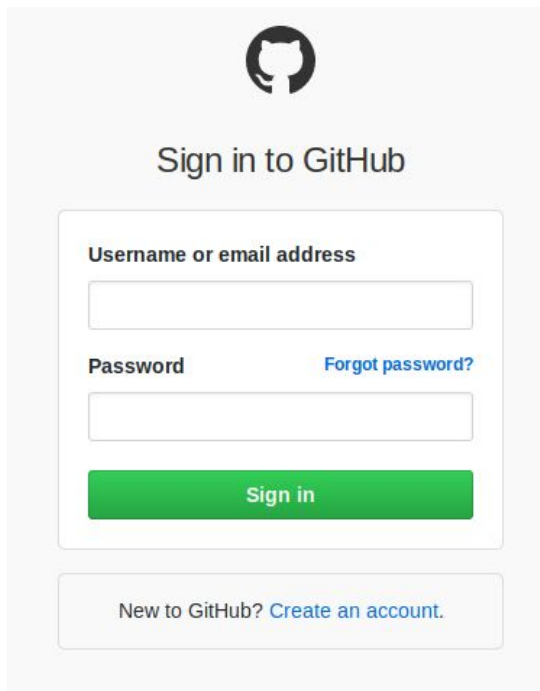
Exercice 1, utilisation d'un dépôt GIT individuel


Télécharger le tutoriel

- Vous savez déjà faire un git clone ?
 - git clone <https://github.com/AMAP-dev/depot-maia.git>
- Sinon allez sur <https://github.com/AMAP-dev/depot-maia> et cliquer sur télécharger le dépôt.
- Le tutoriel se trouve dans
`depot_maia/atelier_git/atelier-maia-git_exercices.pdf`

Connexion à son espace personnel sur Github

<https://github.com/login>

The image shows the GitHub login interface. At the top is the GitHub Octocat logo. Below it is the text 'Sign in to GitHub'. The main form has two input fields: 'Username or email address' and 'Password'. To the right of the password field is a link 'Forgot password?'. Below the password field is a green 'Sign in' button. At the bottom of the form is a link 'New to GitHub? Create an account.'.



Sign in to GitHub

Username or email address

Password [Forgot password?](#)

Sign in

New to GitHub? [Create an account.](#)

Création d'un dépôt distant sur Github

<https://github.com/new>


Créer un nouveau dépôt privé
“premier-depot” par exemple et
cocher l’option “README”.

Create a new repository

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

Owner

Repository name *

 AMAP-dev ▾ / premier-depot ✓

Great repository names are short and memorable. Need inspiration? How about **cautious-barnacle**?

Description (optional)



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** ▾ ⓘ

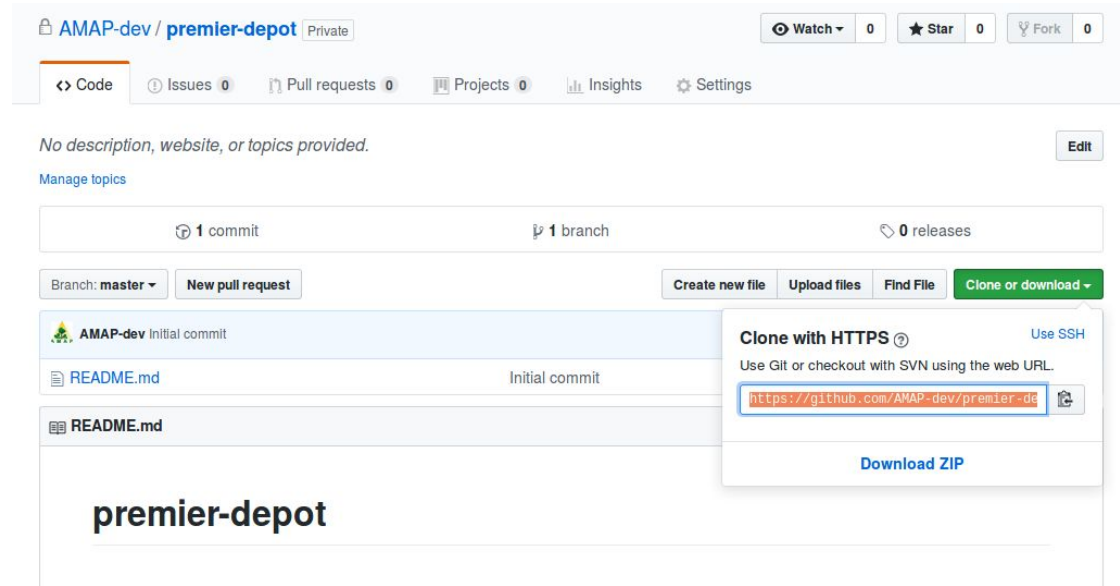
Create repository

Cloner le dépôt distant en local

Ouvrir un invite de commande GIT en local.

Invite de commande GIT

```
git clone https://github.com/votre-login-github/premier-depot.git
```



The screenshot shows the GitHub interface for a repository named 'premier-depot' under the user 'AMAP-dev'. The repository is private. It has 0 watches, 0 stars, and 0 forks. The 'Code' tab is selected, showing a file tree with 'README.md' as the only file. A modal dialog is open, titled 'Clone with HTTPS', providing the URL 'https://github.com/AMAP-dev/premier-depot' and a 'Download ZIP' button.



Programme de la matinée

- **09h00-09h45** : Présentation introductive
- **09h45-10h00** : Pause
- **10h00-11h00** : Exercice 1, travail sur dépôt GitHub personnel, commandes de base en mode “invite de commande”
- **11h00-11h15** : Démonstration exercice 1 avec interface graphique
- **11h15-12h00** : Au choix de chacun
 - Reprise Exercice 1 en mode “interface graphique”
 - Exercice 2, gestion d’un conflit
 - Exercice 3, travail en binôme sur dépôt GitHub partagé



UMR
botanique et Modélisation de l'Architecture des
Plantes et des végétations

Introduction à GIT

Atelier MAIA P3M 04 avril 2019

Nicolas B., François G., Philippe V., Rémi V., Ghislain V.



cirad



INRA
SCIENCE & IMPACT



UNIVERSITÉ
DE MONTPELLIER



Cycle de vie d'un fichier

- De la naissance au commit

