Table des matières

Tutoriel : installer et utiliser Julia, Git/GitHub et VS Code (de zéro)

Ce tutoriel part de zéro. Il s'adresse à une personne qui n'a jamais installé ces outils. Il couvre **Windows**, **macOS** et **Linux**. Suivez la section qui correspond à votre système, puis les sec-

tions communes.

Table des matières

---

1. Vue d’ensemble

Trois outils, trois rôles :

Outil	Rôle
Julia	Langage de programmation pour le calcul scientifique et numérique.
Git	Système de gestion de versions (suivi de l’historique du code, en local).
GitHub	Service en ligne hébergeant des dépôts Git (sauvegarde, partage, collaboration).
VS Code	Éditeur de code qui réunit tout : édition, exécution Julia, et interface Git/GitHub.

> **Git ≠ GitHub.** Git est le logiciel installé sur votre machine. GitHub est un site web qui héberge vos dépôts Git. On peut utiliser Git sans GitHub.

---

2. Installer Julia

La méthode **recommandée aujourd’hui** est `juliaup`, le gestionnaire officiel de versions de Julia. Il installe Julia, gère les mises à jour et permet d’avoir plusieurs versions en parallèle.

Windows

Ouvrez **PowerShell** ou le **Microsoft Store** :

- **Option simple** : cherchez « Julia » dans le Microsoft Store et installez-le (cela installe `juliaup`).
- **Option ligne de commande** : dans PowerShell, tapez :

  ```
  winget install julia -s msstore
  ```

macOS

Dans le **Terminal** :

```
curl -fsSL https://install.julialang.org | sh
```

(Vous pouvez aussi utiliser Homebrew : `brew install juliaup`.)

Linux

Dans un terminal :

```
curl -fsSL https://install.julialang.org | sh
```

Suivez les instructions à l'écran. Fermez puis rouvrez votre terminal à la fin pour que le `PATH` soit mis à jour.

Vérifier l'installation

Dans un terminal **neuf**, tapez :

```
julia --version
```

Vous devriez voir quelque chose comme `julia version 1.11.x`.

Commandes juliaup utiles

```
juliaup status          # versions installées
juliaup update          # mettre à jour Julia
juliaup add lts         # installer la version "long-term support"
juliaup default release  # choisir la version par défaut
```

---

3. Premiers pas avec Julia

Le REPL

Lancez Julia en tapant `julia` dans un terminal. Vous obtenez le **REPL** (invite interactive) :

```
julia> 1 + 1
2

julia> println("Bonjour, Julia !")
Bonjour, Julia !
```

Pour quitter : `exit()` ou `Ctrl-D`.

Les quatre modes du REPL

Tapez ces caractères en début de ligne pour changer de mode :

Touche	Mode	Usage
(défaut)	**Julia**	Exécuter du code.
]	**Pkg**	Gérer les paquets (`add`, `rm`, `status`).
?	**Help**	Afficher l'aide d'une fonction.
;	**Shell**	Exécuter une commande système.

Appuyez sur **Retour arrière** sur une ligne vide pour revenir au mode Julia.

Installer un paquet

Passez en mode Pkg avec `]`, puis :

```
(@v1.11) pkg> add Example
(@v1.11) pkg> status
```

Revenez en mode Julia (Retour arrière) et utilisez-le :

```
julia> using Example
julia> hello("monde")
"Hello, monde"
```

Exécuter un fichier script

Créez un fichier `bonjour.jl` :

```
# bonjour.jl
for i in 1:3
    println("Itération ", i)
end
```

Exécutez-le depuis le terminal :

```
julia bonjour.jl
```

Environnements de projet (bonne pratique)

Pour qu'un projet ait ses propres dépendances, créez/activez un environnement dans le dossier du projet :

```
(@v1.11) pkg> activate .
(@v1.11) pkg> add DataFrames
```

Cela crée deux fichiers, `Project.toml` et `Manifest.toml`, qui décrivent exactement les paquets utilisés. **Versionnez `Project.toml`** avec Git pour que le projet soit reproductible.

---

4. Installer et configurer Git

Installation

Windows : téléchargez « Git for Windows » sur https://git-scm.com/download/win puis lancez l'installateur (les options par défaut conviennent). Cela installe aussi **Git Bash**, un terminal pratique.

macOS :

```
brew install git        # avec Homebrew
# ou simplement :
git --version           # propose d'installer les Command Line Tools
```

Linux (Debian/Ubuntu) :

```
sudo apt update && sudo apt install git
```

Linux (Fedora) : `sudo dnf install git`

Vérifier

```
git --version
```

Configuration initiale (à faire une seule fois)

Indiquez votre identité (elle apparaîtra dans chaque commit) :

```
git config --global user.name "Votre Nom"
git config --global user.email "fabian.bastin@gmail.com"
```

Quelques réglages confortables :

```
git config --global init.defaultBranch main   # branche principale "main"
git config --global pull.rebase false          # stratégie de fusion par défaut
git config --global core.editor "code --wait"  # éditer les messages dans VS Code
```

Vérifiez :

```
git config --list
```

---

5. Créer un compte GitHub et s'authentifier

Créer le compte

1. Allez sur https://github.com et cliquez sur **Sign up**.
2. Choisissez un nom d'utilisateur, votre email, un mot de passe.
3. Activez l'**authentification à deux facteurs (2FA)** : c'est désormais obligatoire pour contribuer.

S'authentifier depuis votre machine

Depuis 2021, GitHub n'accepte plus le mot de passe pour les opérations Git en ligne de commande. Deux méthodes courantes :

Méthode A — HTTPS + Personal Access Token (la plus simple)

1. Sur GitHub : **Settings → Developer settings → Personal access tokens → Tokens (classic) → Generate new token**.
2. Cochez au minimum la portée **`repo`**. Copiez le token (il ne sera plus affiché ensuite).
3. La première fois que vous ferez `git push`, Git demandera identifiant + mot de passe : entrez votre nom d'utilisateur, et **collez le token comme mot de passe**.
4. Pour ne pas le retaper, installez un gestionnaire d'identifiants :
   - Windows : déjà inclus (Git Credential Manager).
   - macOS : `git config --global credential.helper osxkeychain`
   - Linux : `git config --global credential.helper "cache --timeout=3600"`

Méthode B — SSH (pratique sur le long terme)

1. Générez une clé :

   ```
   ssh-keygen -t ed25519 -C "fabian.bastin@gmail.com"
   ```

   Appuyez sur Entrée pour accepter l'emplacement par défaut.

2. Affichez la clé **publique** :

   ```
   cat ~/.ssh/id_ed25519.pub
   ```

3. Copiez-la, puis sur GitHub : **Settings → SSH and GPG keys → New SSH key**, collez-la.
4. Testez :

   ```
   ssh -T git@github.com
   ```

   Vous devriez voir : `Hi <nom>! You've successfully authenticated...`

   Avec SSH, utilisez les URL de dépôt en `git@github.com:...`. Avec HTTPS, utilisez `https://github.com/...`.

---

6. Workflow Git/GitHub de base

Concepts clés

- **Dépôt (repository)** : dossier suivi par Git.
- **Commit** : une « photo » de votre code à un instant donné, avec un message.
- **Branche** : une ligne de développement parallèle (`main` par défaut).
- **Remote** : une copie distante du dépôt (sur GitHub), nommée `origin`.
- Cycle : `modifier → add → commit → push`.

Scénario 1 : démarrer un projet localement et l'envoyer sur GitHub

1. Créez le dépôt **vide** sur GitHub (bouton **New**, sans README pour simplifier).
2. En local :

   ```
   mkdir mon-projet && cd mon-projet
   git init
   echo "# Mon projet" > README.md
   git add README.md
   git commit -m "Premier commit"
   git branch -M main
   git remote add origin https://github.com/<utilisateur>/mon-projet.git
   git push -u origin main
   ```

Scénario 2 : récupérer un projet existant

```
git clone https://github.com/<utilisateur>/mon-projet.git
cd mon-projet
```

Le cycle quotidien

```
git status              # voir les fichiers modifiés
git add fichier.jl      # ajouter à la "staging area" (ou: git add .)
git commit -m "Décrire le changement"
git push                # envoyer sur GitHub
git pull                # récupérer les changements distants
```

Travailler avec des branches

```
git switch -c nouvelle-fonction   # créer + basculer sur une branche
# ... travail, commits ...
git push -u origin nouvelle-fonction
```

Ensuite, sur GitHub, ouvrez une **Pull Request** pour proposer la fusion dans `main`.

Le fichier `.gitignore`

Empêche de versionner certains fichiers. Pour un projet Julia, un bon début :

```
# Julia
/Manifest.toml      # facultatif : à versionner si reproductibilité exacte voulue
*.jl.cov
*.jl.*.cov
/docs/build/
.DS_Store
```

> Pour une **application** ou un travail reproductible, versionnez **aussi** `Manifest.toml`.
> Pour un **paquet réutilisable**, on l'ignore souvent.

---

7. Installer et utiliser VS Code

Installation

Téléchargez VS Code sur https://code.visualstudio.com et installez-le.

- **Windows** : pendant l'installation, cochez « Ajouter à PATH » et « Ouvrir avec Code » (menu contextuel).
- **macOS** : glissez l'app dans Applications, puis dans VS Code, ouvrez la palette de commandes (`Cmd+Shift+P`) → « Shell Command: Install 'code' command in PATH ».
- **Linux** : paquet `.deb`/`.rpm` ou via le gestionnaire de paquets.

Repères de l'interface

- **Barre latérale gauche** : Explorateur de fichiers, Recherche, Source Control (Git), Extensions, Debug.
- **Palette de commandes** : `Ctrl+Shift+P` (`Cmd+Shift+P` sur Mac) — tout est accessible ici.
- **Terminal intégré** : `Ctrl+ò` (ou menu *Terminal* → *New Terminal*).

Installer une extension

Icône **Extensions** (carrés) dans la barre de gauche → cherchez par nom → **Install**.

---

8. Intégration Julia + VS Code

Installer l'extension Julia

1. Ouvrez **Extensions**, cherchez **« Julia »** (éditeur : julialang).
2. Cliquez **Install**.

L'extension détecte normalement Julia automatiquement (grâce à `juliaup`). Sinon, réglez le chemin : `Ctrl+,` → cherchez `julia executable path` → indiquez le chemin de l'exécutable Julia.

Fonctionnalités principales

- **Coloration syntaxique, autocomplétion, info au survol.**
- **REPL Julia intégré** : `Ctrl+Shift+P` → *Julia: Start REPL* (ou `Alt+J Alt+O`).
- **Exécuter une ligne / sélection** : `Shift+Entrée` (envoie au REPL).

- **Exécuter tout le fichier** : `Ctrl+Shift+P` → *Julia: Execute File*, ou le bouton ▶ en haut à droite.
- **Explorateur de variables (workspace)** et **tracé des graphiques (Plots)** dans des panneaux dédiés.
- **Débogueur** : points d'arrêt cliquables dans la marge, exécution pas à pas.

Essai rapide

1. Ouvrez le dossier de votre projet : *File* → *Open Folder*.
2. Créez `essai.jl` :

```
x = collect(1:10)
somme = sum(x)
println("La somme vaut ", somme)
```

3. Placez le curseur sur une ligne et faites `Shift+Entrée` : la ligne s'exécute dans le REPL intégré.

Sélectionner l'environnement de projet

En bas de la fenêtre VS Code, l'extension Julia affiche l'environnement actif (ex. `Julia env: v1.11`). Cliquez dessus pour choisir l'environnement de votre projet (celui qui contient `Project.toml`). C'est l'équivalent de `activate ..`

---

9. Intégration Git/GitHub + VS Code

VS Code intègre Git nativement, plus une extension pour GitHub.

Source Control (Git intégré)

1. Ouvrez l'onglet **Source Control** (icône de branche, barre de gauche) ou `Ctrl+Shift+G`.
2. Les fichiers modifiés apparaissent. Cliquez sur **+** pour les *stager*.
3. Écrivez un **message de commit** en haut, puis cliquez sur le bouton **Commit** (icône en forme de coche).
4. Cliquez sur **Sync Changes** (ou les flèches en bas) pour `push`/`pull`.

Vous voyez aussi : - La **branche courante** en bas à gauche (cliquez pour en changer/créer). - Les **différences** (diff) en cliquant sur un fichier modifié. - Les marges colorées indiquant les lignes ajoutées/modifiées.

Extension GitHub Pull Requests

1. Installez l'extension **« GitHub Pull Requests »** (éditeur : GitHub).
2. Connectez-vous à GitHub via la fenêtre qui s'ouvre (ou *Accounts*, icône en bas à gauche).
3. Vous pouvez alors **créer/relire des Pull Requests** et gérer les **issues** directement dans VS Code.

Publier un projet local sur GitHub en un clic

Si votre dossier n'est pas encore un dépôt : onglet **Source Control** → **Publish to GitHub**. VS Code crée le dépôt distant et pousse le code pour vous (public ou privé au choix).

---

10. Mini-projet de bout en bout

Mettons tout bout à bout.

```
# 1. Créer le projet
mkdir calcul-stats && cd calcul-stats
git init
```

Dans VS Code (*Open Folder* sur `calcul-stats`), ouvrez le REPL Julia et créez l'environnement :

```
(@v1.11) pkg> activate .
(@v1.11) pkg> add Statistics
```

Créez `stats.jl` :

```
using Statistics

donnees = [4, 8, 15, 16, 23, 42]
println("Moyenne : ", mean(donnees))
println("Écart-type : ", std(donnees))
```

Exécutez-le (`Shift+Entrée` ligne par ligne, ou bouton ▶).

Créez un `.gitignore` (voir section 6), puis versionnez :

```
git add .gitignore Project.toml Manifest.toml stats.jl
git commit -m "Projet de statistiques de base"
```

Publiez sur GitHub via **Source Control → Publish to GitHub**, ou en ligne de commande :

```
git branch -M main
git remote add origin https://github.com/<utilisateur>/calcul-stats.git
git push -u origin main
```

Félicitations : vous avez un projet Julia versionné, hébergé sur GitHub, et piloté depuis VS Code.

---

11. Dépannage (FAQ)

`julia: command not found` après installation. Fermez et rouvrez le terminal. Si le problème persiste, le dossier de `juliaup` (`~/.juliaup/bin`) n'est pas dans le `PATH` — relancez l'installateur ou ajoutez-le manuellement.

`git push` est refusé / demande un mot de passe en boucle. GitHub n'accepte plus le mot de passe du compte. Utilisez un **Personal Access Token** (méthode A) ou **SSH** (méthode B), section 5.

VS Code ne trouve pas Julia. `Ctrl+,` → réglage `julia executable path` → pointez vers l'exécutable (visible via `which julia` / `where julia`).

Le REPL Julia de VS Code est lent au premier lancement. Normal : Julia précompile les paquets la première fois. Les fois suivantes sont rapides.

« fatal: not a git repository ». Vous n'êtes pas dans un dossier suivi par Git. Faites `git init`, ou placez-vous dans le bon dossier.

Conflit de fusion (merge conflict). VS Code surligne les zones en conflit avec des boutons *Accept Current / Incoming / Both*. Choisissez, sauvegardez, puis `git add` + `git commit`.

Pour aller plus loin

- Documentation Julia : https://docs.julialang.org
- Pkg.jl (gestion des paquets) : https://pkgdocs.julialang.org
- Pro Git (livre gratuit, en français) : https://git-scm.com/book/fr/v2
- Extension Julia pour VS Code : https://www.julia-vscode.org
- GitHub Docs : https://docs.github.com/fr