Cheat sheet outils

IDE, git, UV, …

Tristan Margaté

2026-01-05

VScode

Téléchargement

1. Rendez-vous sur le site officiel de VScode : https://code.visualstudio.com/download
2. Télécharger et installer la version stable correspondant à votre OS.

Extensions indispensables

Dans l'onglet extensions de VScode (Ctrl+Shift+X), installer les extensions suivantes :

1. Python : Indispensable pour l'autocomplétion et le debugging
2. Jupyter : Pour exécuter et visualiser vos Notebooks dans l'éditeur.
3. Error Lens (recommandé) : Pour visualiser les erreurs dans vos fichiers.
4. Path Intellisense (recommandé) : Pour les suggestions de chemin de fichiers.

> **Command Palette**
>
> Le moteur de recherche interne est accessible par Ctrl+Shift+p. En général vous tapez dedans ce que vous voulez faire, et VScode trouvera la commande.

Pycharm

Installation recommandée :

- Au lieu de télécharger Pycharm directement, télécharger et installer l'application JetBrains Toolbox : https://www.jetbrains.com/toolbox-app/
- Ouvrez l'application JetBrains Toolbox et installer Pycharm.

Avantage de passer par la toolbox :

1. Mise à jour de l'IDE en 1 clic (pas besoin de tout réinstaller)
2. Si une MAJ bug, vous pouvez revenir à une version précédente facilement.

Extensions à installer:

Aucune. Pycharm est déjà très bien configuré pour Python.

> **Search Everywhere**
>
> Appuyez deux fois sur la touche maj (shift + shift). Cela cherche dans tout le projet : noms de fichiers, classes, fonctions, etc. C'est le raccourci ultime

Installer Git

Installation

- Suivez le guide en fonction de votre OS sur https://git-scm.com/book/fr/v2/Démarrage-rapide-Installation-de-Git

Configuration identité

Ouvrez votre terminal et tapez ces deux commandes :

```
1 git config --global user.name "votre_nom"
2 git config --global user.email "votre_email"
```

Vous pouvez vérifier votre identité en tapant

```
1 git config --list
```

En mettant cette identité, Git appose un “tampon” sur vos modifications. Cela sert juste à identifier qui a fait quoi.

> **Identité**
>
> Je vous conseille d’utiliser la même adresse email que votre compte github. Autrement, sur github vos modifications seront signé par un utilisateur “fantôme” non identifié.

Utiliser Git

Commande	Action Technique
`git clone <url>`	Télécharge une copie complète d'un projet distant.
`git pull`	Récupère les mises à jour du serveur.
`git status`	Affiche l'état des fichiers (modifiés, ajoutés…).
`git add <file>`	Place un fichier dans la zone de préparation (*Staging*).
`git commit -m "..."`	Enregistre une version validée de la zone de préparation.
`git push`	Envoie vos commits vers GitHub.

Workflow classique :

- git pull
- git add . (le point signifie : tous les fichiers de mon dossier)
- git commit -m “message”
- git push

Installer UV

Pour installer UV sur votre ordinateur, vous pouvez ouvrir un terminal faire la commande suivante :

Pour Windows :

```
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"
```

Pour macOS et Linux :

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

https://docs.astral.sh/uv/getting-started/installation/

Utiliser UV (les commandes)

Voici les 4 commandes fondamentales pour gérer le cycle de vie de votre projet.

Commande	Action Technique
`uv init`	Initialise un projet (crée `pyproject.toml`)
`uv add <lib>`	Installe une librairie et l'ajoute au config
`uv run <file>`	Exécute le script dans l'environnement isolé
`uv sync`	Installe l'environnement exact (`uv.lock`)

La commande pro : Le src layout

```
uv init --lib  --python 3.13 nom_du_projet
```

Va automatiquement générer le Src layout et préparer le pyproject.toml pour que le code soit installable.

Pour exécuter la fonction main du fichier main.py dans cette nouvelle structure, il suffira d'ajouter dans le pyproject.toml :

```
[project.scripts]
nom-du-projet = "nom_du_projet.main:main"
```

et de faire la commande

```
uv run nom-du-projet
```

Structure projet avec UV

```
nom_du_projet/
├── .git/                # (Dossier caché) Initialisé automatiquement par uv
├── .gitignore           # Liste des fichiers à ignorer par Git
├── .python-version      # Contient juste "3.13" pour forcer la version
├── pyproject.toml       # Le fichier de configuration principal
├── README.md            # Le fichier de présentation (Markdown)
└── src/
    └── nom_du_projet/   # Votre package (les tirets - deviennent des underscores
        └── __init__.py  # Le point d'entrée du package
```

Structure projet UV package

- pyproject.toml : Fichier de configuration central. Il définit l'identité du projet et liste les dépendances directes (ex: pandas) sans fixer leur version.
- .python-version : Fichier indiquant la version de Python utilisée.
- uv.lock : Fichier généré automatiquement lorsque vous importez une librairie (uv add numpy) garantissant la reproductibilité absolue. Il fige les versions exactes de toutes les dépendances et sous-dépendances.
- src/ : Dossier racine isolant le code source de la configuration. Il force une structure professionnelle et évite les conflits d'importation.
- nom_du_projet/ : Le package Python effectif contenant votre logique métier. La présence du fichier **init**.py rend ce dossier importable comme une librairie.

Le pyproject.toml

pyproject.toml

```toml
[project]
name = "clinical-tool"
version = "0.1.0"
description = "Add your description here"
readme = "README.md"
requires-python = ">=3.13"
dependencies = [
    "matplotlib>=3.10.8",
    "numpy>=2.3.5",
    "pandas>=2.3.3",
    "seaborn>=0.13.2",
]

[project.scripts]
clinical-tool = "clinical_tool.main:main"

[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"

[tool.hatch.build.targets.wheel]
packages = ["src/clinical_tool"]

[tool.hatch.build.targets.wheel.sources]
"src" = ""
```

Figure 1: Contenu du pyproject.toml

> ⚠ **Dependencies**
>
> Ne modifiez jamais manuellement les dépendances dans le pyproject.toml. Lorsque vous utilisez uv add library, cela se rajoute automatiquement dans les dépendances.

Premier projet github

Identification github

- Il faut “connecter” sa machine (ordi local) au compte github pour pouvoir mettre en relation les deux
- La méthode la plus propre et rapide est la clé SSH.

1. Créer une clé SSH sur votre laptop, ouvrez votre terminal et faite la commande

```
ssh-keygen -t ed25519 -C "laptop"
```

2. Aller dans votre dossier caché .ssh et copier votre clef publique (fichier id_ed25519.pub) et coller dans une nouvelle clé SSH sur votre compte github (settings/SSH and CPG keys).
3. Aller sur votre compte github et créer un repository vide, avec un nom *name_repo*
4. Sur votre laptop, ouvrez le terminal dans votre projet git (votre dossier crée par UV) et faites la commande

```
git remote set-url origin git@github.com:nom_github/name_repo.Git
```

(attention remplacez bien nom_github par votre pseudo github et name_repo par votre nom de repo)

5. Lors de votre premier push git, il faudra faire la commande

```
git push -u origin main
```