Absolument. C'est une excellente question, et clarifier cette architecture est la clé pour bien utiliser l'outil. Vous touchez au cœur de la différence entre ce qui est sauvegardé et ce qui ne l'est pas.

Voici une explication de l'architecture, des processus de sauvegarde et de restauration, et surtout, de leurs incidences.

**L'Architecture : Git (le logiciel) vs. GitHub (le site web)**

Pour bien comprendre, il faut d'abord distinguer deux choses que l'on confond souvent : **Git** et **GitHub**.

* **Git, c'est le "Moteur"** : C'est un logiciel installé **sur votre ordinateur**. Son unique rôle est de fonctionner comme une "machine à remonter le temps" pour un dossier de projet. Il prend des "photographies" (appelées **commits**) de vos fichiers à un instant T et conserve l'historique de toutes ces photos. Tout ce processus se passe **en local**.
* **GitHub, c'est le "Cloud"** : C'est un service **en ligne** (un site web) qui héberge vos projets Git. C'est une sorte de "Google Drive" ou de "Dropbox" spécialisé pour les projets Git. Il vous permet de stocker une copie de votre projet et de son historique en ligne.

L'interaction entre les deux est simple : vous travaillez sur vos fichiers en local avec Git, et de temps en temps, vous envoyez une copie de votre travail (vos nouveaux "commits") vers GitHub pour la mettre en sécurité et la partager.

**Dans Visual Studio Code, l'onglet "Source Control" 🕊️ est votre interface pour parler à Git.**

**Comment on fait la "Sauvegarde" ? (Commit & Push)**

Votre terme de "sauvegarde" correspond en réalité à un processus en deux ou trois étapes dans le monde de Git. Ce n'est pas juste un bouton "sauvegarder".

1. **Étape 1 : Indexer les changements (Staging)**
   * **Quoi ?** Vous choisissez précisément quels fichiers modifiés vous voulez inclure dans votre prochaine "photographie".
   * **Pourquoi ?** Parfois, vous avez modifié 10 fichiers, mais seulement 5 sont prêts à être sauvegardés. Cela vous donne un contrôle total.
   * **Dans VS Code :** C'est quand vous cliquez sur le + à côté d'un fichier dans la liste "Changes".
2. **Étape 2 : Créer la "photographie" (Commit)**
   * **Quoi ?** Vous prenez la photo des fichiers que vous avez indexés. Chaque commit est un point de sauvegarde permanent dans votre historique local. Il est obligatoire de lui donner un message descriptif (ex : "Ajout de la fonction de calcul des scores").
   * **Dans VS Code :** C'est quand vous tapez votre message dans la case en haut et que vous cliquez sur la coche (✓ Commit).
3. **Étape 3 : Envoyer la copie en ligne (Push)**
   * **Quoi ?** C'est **cette étape** la sauvegarde en ligne ! Vous envoyez tous les commits que vous avez faits en local vers votre dépôt sur GitHub.
   * **Dans VS Code :** C'est le bouton "Sync Changes" ou "Push". Il apparaît après que vous avez fait un commit.

En résumé, la "sauvegarde" complète est le cycle : **Modifier les fichiers → Indexer → Commiter → Pousser vers GitHub**.

**Comment on fait la "Restauration" ? (Le Rollback)**

La restauration d'une version précédente peut se faire de plusieurs manières, de la plus simple (juste regarder) à la plus complexe (annuler des changements).

* **Option 1 : Juste consulter une ancienne version (Checkout)** Vous pouvez dire à Git : "Montre-moi à quoi ressemblaient mes fichiers au moment de ce commit précis". C'est un mode lecture seule, idéal pour copier un bout de code que vous aviez supprimé.
* **Option 2 : Créer une branche à partir d'une ancienne version (Branch)** C'est la méthode la plus sûre. Vous créez une nouvelle ligne temporelle à partir d'une ancienne version pour travailler dessus sans affecter votre projet principal. C'est ce que nous ferons pour le script

rollback\_gas.ps1.

* **Option 3 : Annuler un commit existant (Revert)** Cette commande ne supprime pas le passé. Elle crée un **nouveau commit** qui fait l'exact opposé des changements d'un commit précédent. C'est propre et ça ne réécrit pas l'historique.

**Quelles sont les incidences de la restauration ? (La question la plus importante)**

C'est ici qu'il faut être très clair sur ce que Git/GitHub sauvegarde.

**1. Ce que Git sauvegarde : VOS FICHIERS LOCAUX**

Git ne connaît **que les fichiers présents dans votre dossier de projet** sur votre ordinateur. Cela inclut :

* Les scripts .js de Google Apps Script.
* Le fichier appsscript.json de clasp.
* Vos scripts PowerShell .ps1.
* Les **exports CSV** que vous générez à un instant T.

**2. Ce que Git ne sauvegarde PAS : L'ÉTAT EN LIGNE**

Git n'a **aucune connexion directe ou connaissance** de vos services Google :

* Il ne connaît pas la structure actuelle de vos onglets Google Sheets.
* Il ne connaît pas les données qui sont *actuellement* dans vos Google Sheets.
* Il ne connaît pas les déclencheurs (triggers) configurés sur Google Apps Script.

**Conséquences de la restauration :**

* **Sur les scripts (le code)** :
  + Si vous restaurez une ancienne version du code et la "poussez" vers Google Apps Script, vous remplacez le code en ligne.
  + **Incidences** : Comme nous l'avons noté hier, cela peut vous obliger à recréer des déclencheurs, à ré-autoriser des permissions si les scopes ont changé, ou à redéployer une Web App.
* **Sur les fichiers Google Sheets (la structure et les données)** :
  + **C'est le point crucial : Restaurer une version dans Git n'a AUCUN impact direct sur vos fichiers Google Sheets en ligne.**
  + Si la structure des colonnes de votre Google Sheet a changé, Git ne le sait pas. Les seuls "témoins" de l'ancienne structure sont les fichiers CSV que vous avez sauvegardés dans un commit passé.
  + Pour "restaurer" la structure ou les données d'un Google Sheet, il faudrait faire une

**action manuelle** : récupérer l'ancien fichier CSV depuis l'historique Git et le réimporter dans votre Google Sheet. C'est là que des problèmes peuvent survenir si, par exemple, des formules dépendent de colonnes qui n'existent plus.

En conclusion, voyez Git et GitHub comme l'historique de votre **code** et des **fichiers de configuration**. La base de données (Google Sheets) vit sa propre vie et la seule façon d'en garder une trace dans Git est via les exports CSV que nous avons automatisés.