



GitHub Pages- HUGO

SOMMAIRE



1 Git et GitHub

2 HUGO

3 Markdown Language

4 GitHub Pages - Hugo



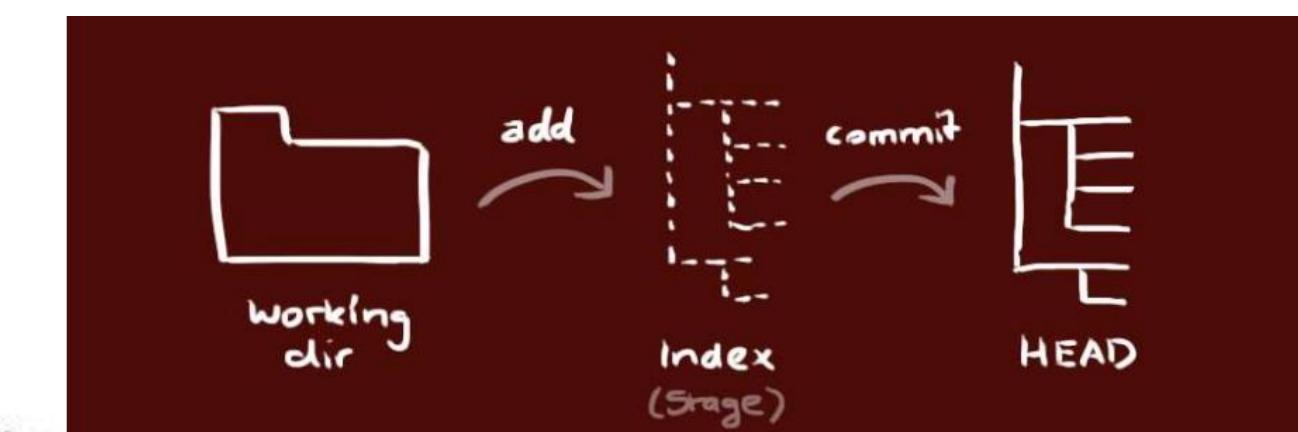
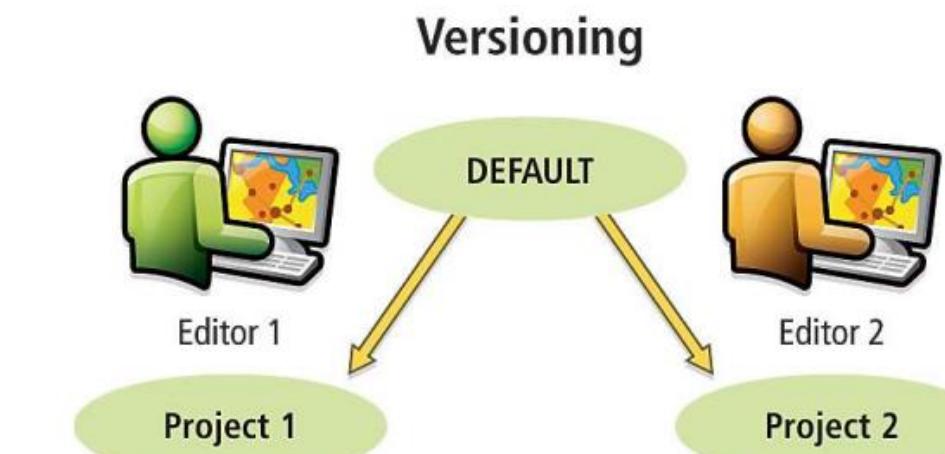
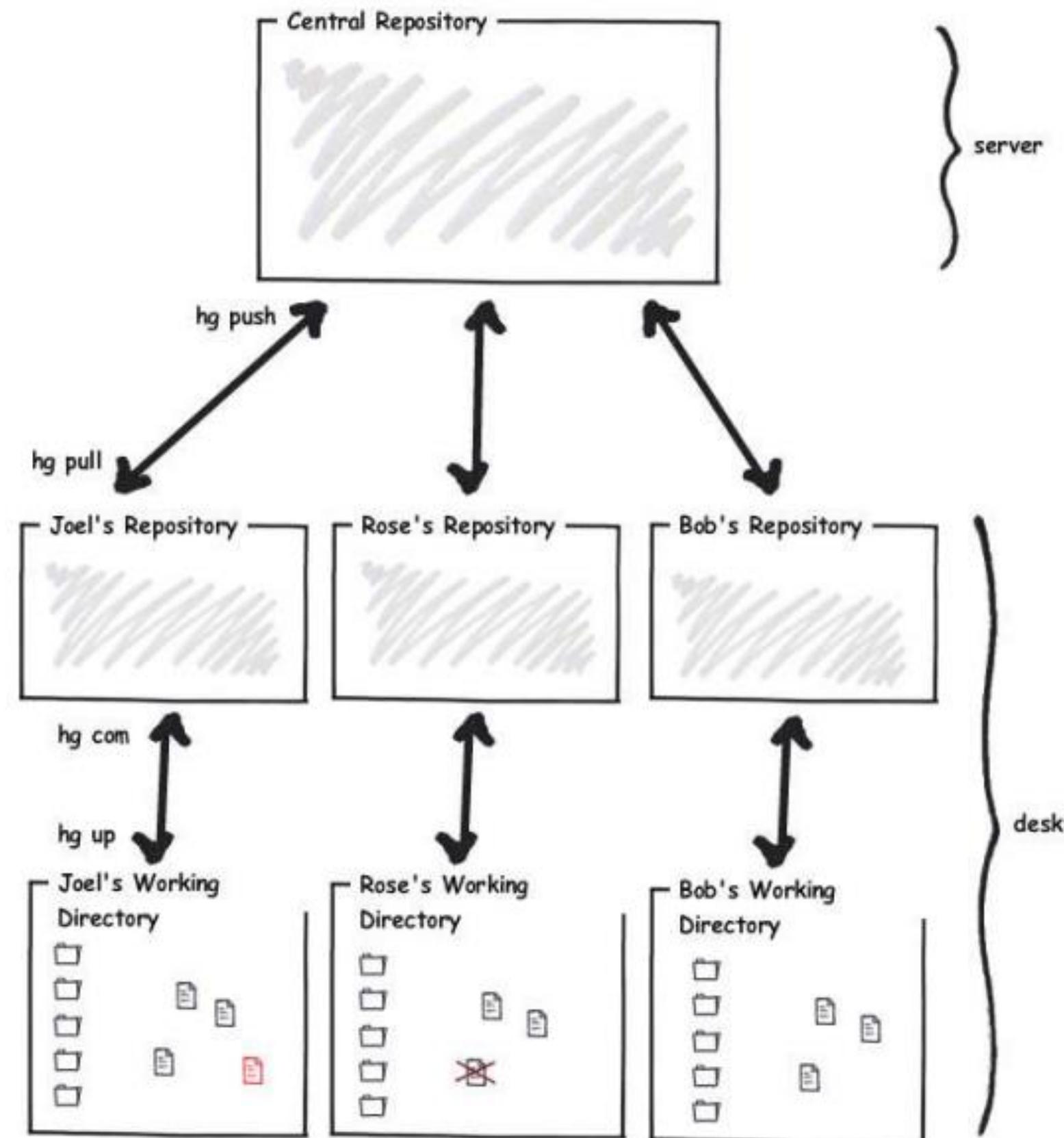
Versioning



Problématique

- Une équipe de développement composée de 3 personnes réalise une application. Ils souhaitent s'échanger leur fichiers sources. Comment procéder ?
- Un développeur a écrit du code qui fait planter l'application. Pire encore, un autre développeur a supprimé des fichiers par mégarde. Comment restaurer les fichiers ?
- 2 développeurs ont modifié le même fichier avec des modifications différentes. Comment les fusionner ?

HO
II
HO





Principales Commandes Git

1. Initialisation d'un référentiel Git :

- `git init`: Initialise un nouveau référentiel Git.

2. Clonage d'un référentiel existant :

- `git clone <URL>` : Clone un référentiel Git existant depuis une URL.

3. Gestion des modifications :

- `git status`: Affiche l'état des fichiers dans le répertoire de travail.
- `git add <fichier>`: Ajoute des modifications de fichiers à la zone de préparation.
- `git reset <fichier>`: Désélectionne les fichiers de la zone de préparation.
- `git diff`: Affiche les modifications non ajoutées.
- `git commit -m "Message de commit"`: Crée un commit avec les modifications en cours.



Principales Commandes Git

4. Historique et commits :

- `git log`: Affiche l'historique des commits.
- `git show <commit>`: Affiche les détails d'un commit spécifique.

5. Branches :

- `git branch`: Affiche la liste des branches.
- `git branch <nom-de-branche>`: Crée une nouvelle branche.
- `git checkout <nom-de-branche>`: Bascule vers une branche spécifique.
- `git merge <nom-de-branche>`: Fusionne une branche dans la branche actuelle.
- `git branch -d <nom-de-branche>`: Supprime une branche.

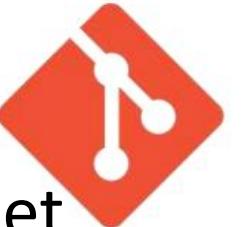
6. Récupération et envoi de modifications :

- `git pull`: Récupère les modifications depuis un référentiel distant et fusionne les modifications.
- `git push`: Envoie les modifications locales vers un référentiel distant.



Commits Conventionnels

Les commits conventionnels sont une pratique recommandée pour structurer les messages de commit dans un projet Git. Ils permettent une meilleure compréhension de l'historique du code et facilitent la collaboration.

**git**

Structure d'un Commit Conventionnel

Un message de commit conventionnel suit généralement la structure suivante :

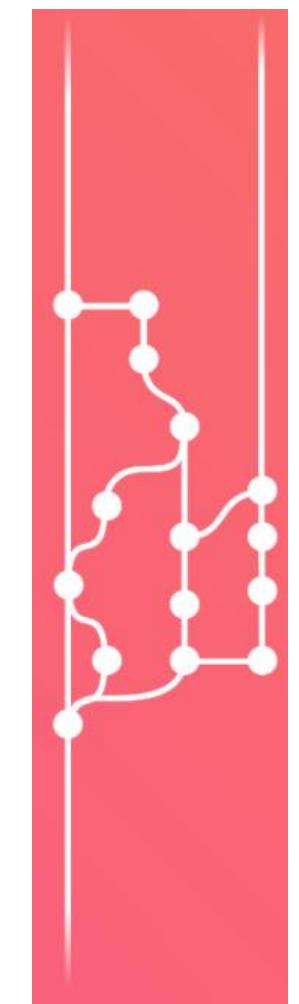
- `<type>` : Indique la nature du changement (ex. : feat, fix, docs, style, refactor, perf, test, chore).
- `<portée>` : Facultatif, indique la portée du changement.
- `<description>` : Une brève description du changement.

Exemple :

feat(users): Ajoute la fonctionnalité de connexion

fix(bugs): Corrige le bug d'affichage sur la page d'accueil

docs(readme): Met à jour la documentation du projet



Avantages :

- Améliore la lisibilité de l'historique.
- Facilite la génération de notes de version automatiques.
- Permet de suivre les types de changements plus précisément.



Guide de commits conventionnel

1. `feat` (Feature):

- Utilité : Ajout d'une nouvelle fonctionnalité au projet.
- Exemple : "feat(users)": Ajoute la fonctionnalité de connexion."

2. `fix` (Bug Fix):

- Utilité : Correction d'un bogue ou d'une erreur dans le code.
- Exemple : "fix(bugs)": Corrige le bug d'affichage sur la page d'accueil."

3. `docs` (Documentation):

- Utilité : Mise à jour de la documentation du projet.
- Exemple : "docs(readme)": Met à jour la documentation du projet.

4. `style` (Styling):

- Utilité : Modifications liées au style, à la mise en forme ou à l'apparence visuelle du code.
- Exemple : "style(css)": Refonte du style de la page de profil."

5. `refactor` (Refactor):

- Utilité : Restructuration du code sans ajouter ni supprimer de fonctionnalités.

- Exemple : "refactor(api)": Réorganise les endpoints de l'API."

6. `perf` (Performance):

- Utilité : Amélioration des performances du code.
- Exemple : "perf(database)": Optimise les requêtes de base de données."

7. `test` (Tests):

- Utilité : Ajout ou modification des tests unitaires ou fonctionnels.
- Exemple : "test(auth)": Ajoute des tests pour le système d'authentification.

8. `chore` (Chores):

- Utilité : Tâches de maintenance, mises à jour de dépendances.
- Exemple : "chore(deps)": Met à jour les dépendances du projet."



1^{er} étape. Installation GitBash :

<https://git-scm.com/downloads>

The screenshot shows the official Git website at <https://git-scm.com/>. The main navigation bar includes links for "About", "Documentation", "Downloads" (which is highlighted in red), "Community", and a link to "The entire Pro Git book". The "Downloads" section features three buttons for "macOS", "Windows", and "Linux/Unix". Below these buttons, a note states: "Older releases are available and the Git source repository is on GitHub." To the right, a large monitor icon displays the "Latest source Release" which is "2.42.1" (Release Notes from 2023-11-02) with a "Download for Windows" button.

2^{ème} étape. configuration GitBash :

`git config --global user.name github_name`

`git config --global user.email user@gmail.com`



What is Hugo?

Hugo est un générateur de sites statiques, un outil qui transforme du contenu en pages web HTML.

Il est rapide, efficace, et repose sur le langage de programmation Go. Contrairement aux systèmes de gestion de contenu dynamiques, Hugo crée des fichiers HTML prêts à être servis aux visiteurs.

C'est idéal pour les sites web plus rapides, plus sécurisés, et plus simples à gérer. Hugo utilise des modèles pour organiser la mise en page de votre site web, et il peut être personnalisé grâce à des thèmes et des fonctionnalités supplémentaires.

Pratique : Guide d'installation et configuration HUGO



1^{er} étape. Installation Hugo : <https://github.com/gohugoio/hugo/releases/tag/v0.120.4>

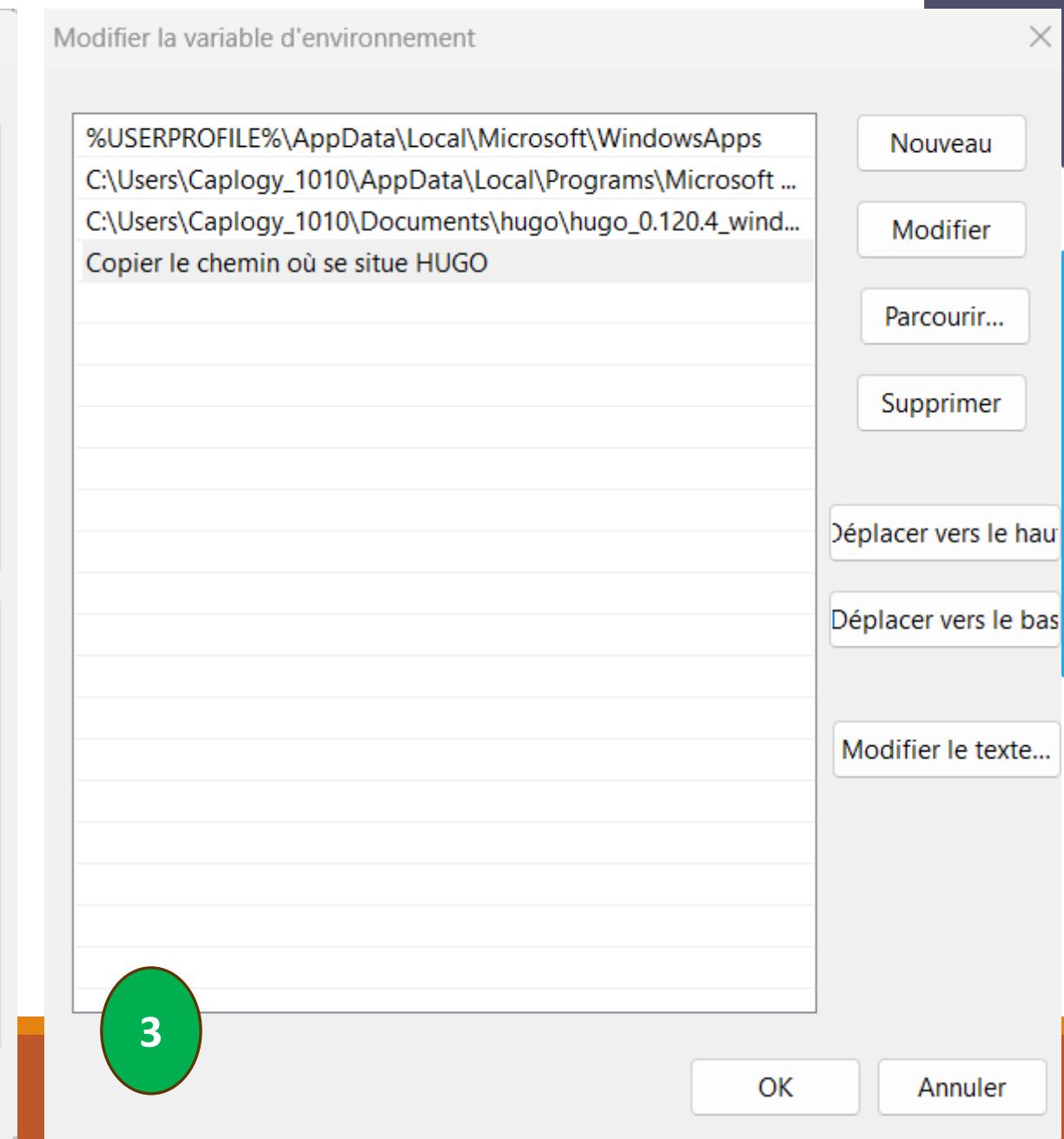
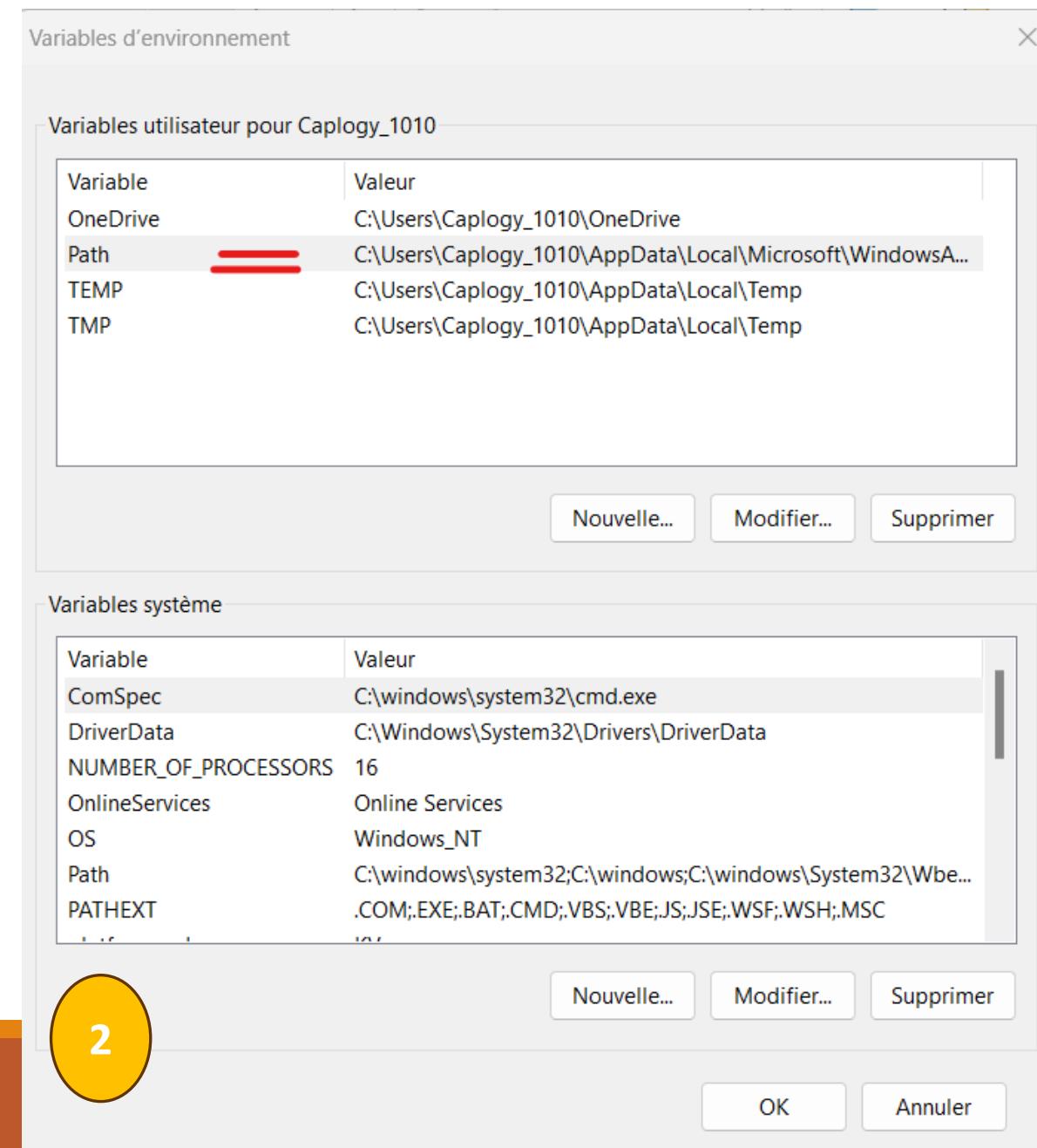
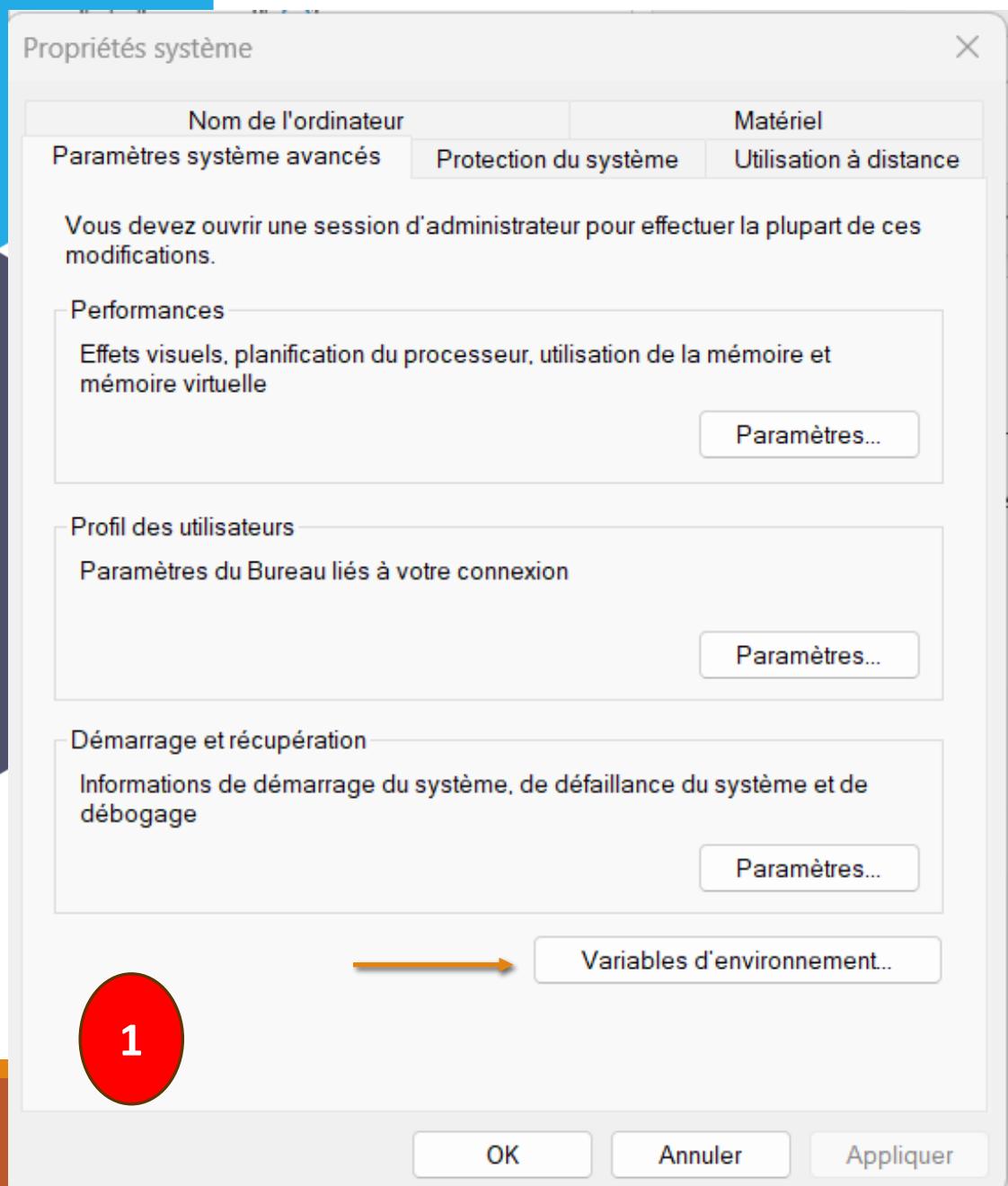
↳hugo_0.120.4_checksums.txt	2.08 KB
↳hugo_0.120.4_darwin-universal.tar.gz	39.9 MB
↳hugo_0.120.4_dragonfly-amd64.tar.gz	19.3 MB
↳hugo_0.120.4_freebsd-amd64.tar.gz	19.3 MB
↳hugo_0.120.4_Linux-64bit.tar.gz	19.3 MB
↳hugo_0.120.4_linux-amd64.deb	20.2 MB
↳hugo_0.120.4_linux-amd64.tar.gz	19.3 MB
↳hugo_0.120.4_linux-arm.tar.gz	17.8 MB
↳hugo_0.120.4_linux-arm64.deb	18.8 MB
↳hugo_0.120.4_linux-arm64.tar.gz	17.9 MB
↳hugo_0.120.4_netbsd-amd64.tar.gz	19.3 MB
↳hugo_0.120.4_openbsd-amd64.tar.gz	19.3 MB
↳hugo_0.120.4_solaris-amd64.tar.gz	18.6 MB
↳hugo_0.120.4_windows-amd64.zip	20.1 MB
↳hugo_0.120.4_windows-arm64.zip	18.4 MB
↳hugo_extended_0.120.4_darwin-universal.tar.gz	41.7 MB
↳hugo_extended_0.120.4_Linux-64bit.tar.gz	20.5 MB
↳hugo_extended_0.120.4_linux-amd64.deb	21.5 MB
↳hugo_extended_0.120.4_linux-amd64.tar.gz	20.5 MB
↳hugo_extended_0.120.4_linux-arm64.deb	19.9 MB
↳hugo_extended_0.120.4_linux-arm64.tar.gz	19 MB
↳hugo_extended_0.120.4_windows-amd64.zip	21.4 MB



2^{ème} étape. Configuration du path Hugo :

Sous Windows :

- Recherchez "Variables d'environnement" dans la barre de recherche Windows.
- Cliquez sur "Modifier les variables d'environnement système."
- Dans la section "Variables système," recherchez la variable "Path" ensuite cliquez sur "Modifier."
- Cliquez sur "Nouveau" et ajoutez le chemin vers le répertoire où Hugo est installé.





Principales Commandes HUGO :



1. Créer un nouveau site Hugo :

- Pour créer un nouveau site Hugo, utilisez la commande :

hugo new site nom-de-votre-site

2. Ajouter un nouveau contenu :

- Pour créer un nouveau contenu (par exemple, un article de blog) :

hugo new nom-du-contenu/titre-de-l-article.md

3. Lancer un serveur de développement :

- Pour prévisualiser votre site localement pendant le développement :

hugo server -D

- L'option `'-D` permet de voir les brouillons (drafts).

4. Lister les commandes et options:

- Pour obtenir une liste complète des commandes et options Hugo :

hugo help



Principales Commandes HUGO :



5. Personnaliser les paramètres de configuration :

- Éditez le fichier `config.toml` pour personnaliser les paramètres de configuration de votre site Hugo.

6. Installer un thème :

- Pour ajouter un thème Hugo à votre site, modifiez le fichier de configuration (`config.toml`) pour spécifier le thème que vous souhaitez utiliser.

7. Créer une nouvelle section :

- Pour ajouter une nouvelle section à votre site (par exemple, "portfolio") :

```
hugo new section/nom-de-la-section/_index.md
```

8. Afficher la version de Hugo :

- Pour afficher la version de Hugo installée :

```
hugo version
```



Markdown language .md :



Markdown est un langage de balisage léger utilisé pour formater du texte de manière simple et lisible. Contrairement à des langages comme HTML ou LaTeX, Markdown est conçu pour être facile à lire et à écrire, même pour les personnes qui ne sont pas des experts en informatique.

1. Texte en gras et en italique :

- Pour mettre un mot en gras, entourez-le de deux astérisques : ****mot en gras****
- Pour mettre un mot en italique, utilisez un seul astérisque : **mot en italique**

2. Titres et sous-titres :

- Vous pouvez créer des titres en utilisant des dièses (#). Par exemple, # Titre 1 pour un titre de niveau 1.
- Pour des sous-titres, ajoutez plus de dièses, par exemple, ## Titre 2 pour un sous-titre de niveau 2.

3. Listes :

- Pour créer une liste à puces, utilisez un astérisque ou un tiret suivi d'un espace : * Élément de liste.
- Pour créer une liste numérotée, utilisez des chiffres suivis d'un point : 1. Premier élément.

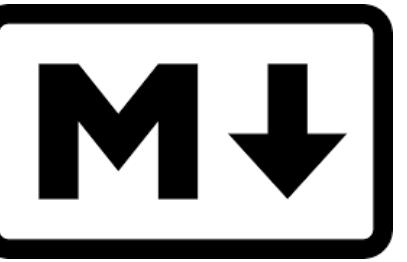
4. Liens :

- Pour ajouter un lien, utilisez la syntaxe [texte du lien](URL)
Par exemple,

[Google](https://www.google.com)



Markdown language .md :



5. Images :

- Pour insérer une image, utilisez ![texte alternatif](URL de l'image)

6. Bloc de code :

- Pour inclure du code, entourez-le de trois backticks (`) pour créer un bloc de code. Vous pouvez spécifier la langue du code après les trois backticks pour obtenir une coloration syntaxique.

7. Citations :

- Pour créer des citations, utilisez le signe supérieur à (>). Par exemple,
> Ceci est une citation.

8. Tableaux :

- Vous pouvez créer des tableaux en utilisant des barres verticales (|) pour séparer les cellules. Les lignes de titre sont séparées par des tirets (-).

9. Notes de bas de page :

- Vous pouvez ajouter des notes de bas de page en utilisant la syntaxe [^1] dans le texte, puis définir la note correspondante en bas du document.



Pratique Mise en route d'un projet HUGO :



```
1 hugo new site demo #suivant la commande de création d'un dossier projet hugo  
2 cd demo #accéder au projet hugo  
3 git init #initialiser un repository suivant le fichier caché .git  
4 git submodule add https://github.com/theNewDynamic/gohugo-theme-ananke.git themes/ananke #ajouter le thème ananke  
5 echo "theme = 'ananke'" >> hugo.toml #mettre en place le thème en modifiant la config .toml  
6 hugo server #lancer le projet hugo en local par défaut https://localhost:1313
```

Lien du thème : [git submodule add https://github.com/theNewDynamic/gohugo-theme-ananke](https://github.com/theNewDynamic/gohugo-theme-ananke)

Rajouter une page sur votre projet :

```
1 hugo new content page/first_page.md
```

Thèmes Hugo (:

<https://master--hugothemes.netlify.app/?search-input=portfolio>



Remplissage du contenu suivant markdown language syntax:

Exemple :



```
## Versionning the right way
After creating and setting up a repo
```bash
git init
git commit --allow-empty -m "⚡ init master branch"
git remote add origin <link to gihtub new repo>
git push -u origin master
```

> till this stage the master branch is set up and ready with an empty commit

__Create & init a develop branch__
```bash
git checkout -b develop # create a new branch called develop = création d'une branche intitulé develop
git commit --allow-empty -m "⚡ init develop branch"
git push origin develop
```

__Branch feature first__
```bash
git checkout -b feature/first
the branch introducing the first feature is created .
make the necessary modifications on your project
git add . # we add all the modification , we must verify in the case of a project the .gitignore file is created .
git commit -m "feat: 🚧 add the project boilerplate"
git push origin feature/first
```

```



GitHub Pages

Introduction à GitHub Pages

- 1 : GitHub Pages est un service d'hébergement web proposé par GitHub.
- 2 : Il vous permet d'héberger vos sites web statiques, votre documentation et d'autres contenus web directement à partir de votre référentiel GitHub.
- 3 : GitHub Pages est un excellent moyen de présenter vos projets open source, vos blogs personnels ou la documentation de votre projet.



GitHub Pages

Mise en Place Github Pages :

- 1 : Créez un nouveau référentiel sur GitHub ou utilisez un référentiel existant.
- 2 : Assurez-vous que votre référentiel contient le projet dans le répertoire racine pour la page principale.
- 3 : Accédez aux Paramètres de votre référentiel, faites défiler jusqu'à la section Pages et choisir l'option du déploiement à partir de github actions.
- 4 : Votre site sera accessible à une URL comme
<https://nom-d'utilisateur.github.io/nom-du-référentiel>



Mise en Place Github Pages sur un projet HUGO

<https://gohugo.io/hosting-and-deployment/hosting-on-github/>



GitHub Pages

- 1/ Fermer tous les dossiers sur l'architecture de votre projet.
- 2/ Cliquer 2 fois ou sélectionner créer un fichier **.github/workflows/main.yaml**
- 3/ Copier le workflow prérequis à partir du lien en haut => **step 6** .
- 4/ Changer la branche Principale
`name: Deploy Hugo site to Pages`
`on: # Runs on pushes targeting the default branch`
`push:`
`branches: - main ←—`
- Changer main par **master**.
- 5/ Dès que vous envoyez un commit sur votre repertoire Github , un workflow sera visible à partir de la rubrique **Actions**.