GestVenv - Documentation des commandes et utilisation dans un projet de développement

Sommaire

1. Introduction
2. Installation
3. Commandes de base
4. Gestion des environnements
5. Gestion des packages
6. Import et export de configurations
7. Utilisation dans un projet de développement
8. Intégration avec d'autres outils
9. Bonnes pratiques
10. Dépannage

1. Introduction

GestVenv est un gestionnaire d'environnements virtuels Python qui simplifie et centralise la gestion des environnements virtuels. Il offre une interface unifiée pour créer, gérer et partager des environnements virtuels, remplaçant la nécessité d'utiliser plusieurs outils comme venv, virtualenv ou pipenv.

Avantages clés

- **Interface unifiée** pour toutes les opérations liées aux environnements virtuels
- **Gestion centralisée** de tous vos environnements
- **Simplification du partage** d'environnements entre développeurs
- **Support multi-versions** de Python
- **Compatible** avec Windows, macOS et Linux

2. Installation

Installation depuis PyPI

```
pip install gestvenv
```

Installation depuis le code source

```
git clone https://github.com/thearchit3ct/gestvenv.git
cd gestvenv
pip install -e .
```

Vérification de l'installation

```
gestvenv --version
```

3. Commandes de base

GestVenv utilise une interface en ligne de commande avec la structure suivante :

```
gestvenv <commande> [options] [arguments]
```

Obtenir de l'aide

```
# Aide générale
gestvenv --help

# Aide sur une commande spécifique
gestvenv create --help

# Documentation détaillée
gestvenv docs
gestvenv docs commandes  # Documentation spécifique aux commandes
```

Liste des commandes principales

Commande	Description
`create`	Crée un nouvel environnement virtuel
`activate`	Active un environnement virtuel
`deactivate`	Désactive l'environnement actif
`delete`	Supprime un environnement virtuel
`list`	Liste tous les environnements virtuels
`info`	Affiche des informations sur un environnement
`install`	Installe des packages dans un environnement
`uninstall`	Désinstalle des packages d'un environnement
`update`	Met à jour des packages dans un environnement
`export`	Exporte la configuration d'un environnement
`import`	Importe une configuration d'environnement
`clone`	Clone un environnement existant
`run`	Exécute une commande dans un environnement

Commande	Description
`config`	Configure les paramètres par défaut
`check`	Vérifie les mises à jour disponibles pour les packages
`pyversions`	Liste les versions Python disponibles sur le système
`docs`	Affiche la documentation

4. Gestion des environnements

Création d'un environnement

```
# Création simple avec Python par défaut
gestvenv create mon_projet

# Spécifier une version Python
gestvenv create mon_projet --python python3.9

# Inclure des packages initiaux
gestvenv create mon_projet --packages "flask,pytest,gunicorn"

# Spécifier un chemin personnalisé
gestvenv create mon_projet --path "/chemin/personnalise/mon_projet"
```

Activation et désactivation

```
# Activer un environnement
gestvenv activate mon_projet
# Suivre les instructions affichées pour compléter l'activation

# Désactiver l'environnement actif
gestvenv deactivate
# Suivre les instructions affichées
```

> **Note :** Contrairement à d'autres gestionnaires d'environnements virtuels, GestVenv affiche les commandes à exécuter pour activer/désactiver l'environnement, car un script Python ne peut pas directement modifier l'environnement du shell parent.

Listing et informations

```
# Lister tous les environnements
gestvenv list

# Afficher des informations détaillées
gestvenv list --verbose
```

```
# Informations complètes sur un environnement spécifique
gestvenv info mon_projet
```

Suppression d'un environnement

```
# Avec confirmation
gestvenv delete mon_projet

# Sans confirmation
gestvenv delete mon_projet --force
```

Clonage d'un environnement

```
gestvenv clone mon_projet mon_projet_dev
```

5. Gestion des packages

Installation de packages

```
# Dans l'environnement actif
gestvenv install "flask,pytest"

# Dans un environnement spécifique
gestvenv install "flask,matplotlib" --env mon_projet

# Spécifier des versions
gestvenv install "flask==2.0.1,pytest>=6.0.0"
```

Désinstallation de packages

```
# Dans l'environnement actif
gestvenv uninstall "flask,pytest"

# Dans un environnement spécifique
gestvenv uninstall "flask" --env mon_projet
```

Mise à jour de packages

```
# Vérifier les mises à jour disponibles
gestvenv check
```

```
# Mettre à jour des packages spécifiques
gestvenv update "flask,pytest"

# Mettre à jour tous les packages
gestvenv update --all

# Dans un environnement spécifique
gestvenv update --all --env mon_projet
```

Exécution de commandes dans un environnement

```
# Exécuter un script Python
gestvenv run mon_projet python script.py

# Exécuter un module
gestvenv run mon_projet python -m pytest

# Exécuter d'autres commandes
gestvenv run mon_projet npm install
```

6. Import et export de configurations

Export de configuration

```
# Au format JSON (par défaut)
gestvenv export mon_projet

# Spécifier un chemin de sortie
gestvenv export mon_projet --output mon_projet_config.json

# Au format requirements.txt
gestvenv export mon_projet --format requirements

# Ajouter des métadonnées
gestvenv export mon_projet --add-metadata "description:Projet Web
Flask,auteur:Alice"
```

Import de configuration

```
# Depuis un fichier JSON
gestvenv import mon_projet_config.json

# Depuis un fichier requirements.txt
gestvenv import requirements.txt --name nouveau_projet
```

```
# Spécifier un nom différent
gestvenv import mon_projet_config.json --name autre_nom
```

7. Utilisation dans un projet de développement

Initialisation d'un nouveau projet

```
# 1. Créer le répertoire du projet
mkdir mon_nouveau_projet
cd mon_nouveau_projet

# 2. Initialiser un dépôt Git (optionnel)
git init

# 3. Créer un environnement virtuel
gestvenv create env --python python3.9

# 4. Activer l'environnement
gestvenv activate env
# Suivre les instructions d'activation

# 5. Installer les packages de base
gestvenv install "flask,pytest,python-dotenv"

# 6. Créer un fichier requirements.txt pour le projet
gestvenv export env --format requirements --output requirements.txt

# 7. Ajouter des fichiers au .gitignore
echo ".env" >> .gitignore
echo "__pycache__/" >> .gitignore
```

Configuration pour le développement et la production

Pour maintenir séparés les environnements de développement et de production :

```
# Créer un environnement de développement
gestvenv create dev_env --python python3.9 --packages
"flask,pytest,python-dotenv,debugpy"

# Créer un environnement de production
gestvenv create prod_env --python python3.9 --packages
"flask,gunicorn,python-dotenv"

# Exporter les configurations
gestvenv export dev_env --format requirements --output requirements-
dev.txt
gestvenv export prod_env --format requirements --output requirements.txt
```

Workflow quotidien de développement

```
# 1. Activer l'environnement
gestvenv activate dev_env
# Suivre les instructions

# 2. Vérifier les mises à jour disponibles
gestvenv check

# 3. Mettre à jour les packages si nécessaire
gestvenv update --all

# 4. Installer de nouveaux packages pour le développement
gestvenv install "black,flake8"

# 5. Exécuter des tests
gestvenv run dev_env pytest

# 6. Mettre à jour le fichier requirements
gestvenv export dev_env --format requirements --output requirements-
dev.txt
```

Collaboration en équipe

Pour faciliter la collaboration entre développeurs :

```
# Développeur 1: Exporter la configuration
gestvenv export dev_env --output projet_config.json --add-metadata
"description:Configuration de développement,équipe:Backend"

# Ajouter au dépôt Git
git add projet_config.json
git commit -m "Ajouter la configuration d'environnement"
git push

# Développeur 2: Récupérer et importer la configuration
git pull
gestvenv import projet_config.json --name dev_env
```

8. Intégration avec d'autres outils

Intégration avec Visual Studio Code

1. Ouvrir le projet dans VS Code
2. Ouvrir la palette de commandes (Ctrl+Shift+P ou Cmd+Shift+P)
3. Sélectionner "Python: Select Interpreter"
4. Choisir l'interpréteur Python de votre environnement GestVenv

Path pour VS Code sur différents OS:

- Windows: `C:\Users\[username]\.gestvenv\environments\[env_name]\Scripts\python.exe`
- macOS/Linux: `~/.config/gestvenv/environments/[env_name]/bin/python`

Intégration avec pytest

```
# Configuration de pytest avec un environnement GestVenv
gestvenv run mon_projet pytest -v tests/
```

Pour configurer un fichier `pytest.ini` :

```
[pytest]
testpaths = tests
python_path = .
```

Intégration avec tox

Exemple de fichier `tox.ini` :

```
[tox]
envlist = py38,py39
isolated_build = True

[testenv]
deps = -r{toxinidir}/requirements-test.txt
commands =
    pytest {posargs:tests}
```

Utilisation avec GestVenv :

```
# 1. Exporter les dépendances de test
gestvenv export test_env --format requirements --output requirements-
test.txt

# 2. Exécuter tox
gestvenv run main_env tox
```

9. Bonnes pratiques

Structure de projet recommandée

```
mon_projet/
├── .git/
├── .gitignore
├── README.md
├── setup.py ou pyproject.toml
├── requirements.txt
├── requirements-dev.txt
├── gestvenv_config.json
├── mon_module/
│   ├── __init__.py
│   └── ...
└── tests/
    ├── __init__.py
    └── ...
```

Versionnage des dépendances

Pour assurer la reproductibilité de l'environnement, fixez les versions dans vos exports :

```
# Exporter avec les versions exactes (recommandé pour production)
gestvenv export prod_env --format requirements --output requirements.txt

# Pour le développement, vous pouvez être plus flexible
gestvenv install "pytest>=6.0.0,black" --env dev_env
```

Gestion des différents environnements

Il est recommandé de maintenir différents environnements pour différentes phases du projet :

1. **dev_env** - Pour le développement, avec des outils de débogage et de test
2. **test_env** - Pour les tests automatisés
3. **prod_env** - Pour la production, avec seulement les dépendances nécessaires

Automatisation avec des scripts

Créez des scripts shell/batch pour automatiser les opérations courantes :

```
#!/bin/bash
# setup_dev.sh

# Configurer l'environnement de développement
gestvenv create dev_env --python python3.9
gestvenv activate dev_env
# ... instructions d'activation affichées ...

# Installer les dépendances
gestvenv install "flask,pytest,black,flake8,python-dotenv"
```

```
# Exporter la configuration
gestvenv export dev_env --format requirements --output requirements-
dev.txt
```

10. Dépannage

Problèmes d'activation

Problème: La commande d'activation ne fonctionne pas.

Solution:

- Assurez-vous d'exécuter la commande exacte affichée par `gestvenv activate`.
- Sur Windows, assurez-vous d'utiliser CMD ou PowerShell comme indiqué.
- Sur Linux/macOS, utilisez `source` comme indiqué.

```
# Exemple correct sur Linux/macOS
source /chemin/vers/env/bin/activate

# Exemple correct sur Windows
C:\chemin\vers\env\Scripts\activate.bat
```

Conflits de packages

Problème: Erreurs lors de l'installation des packages à cause de conflits de dépendances.

Solution:

```
# Vérifier les conflits potentiels
gestvenv info mon_env

# Installer les packages un par un
gestvenv install flask
gestvenv install pytest
```

Versions Python manquantes

Problème: La version Python spécifiée n'est pas disponible.

Solution:

```
# Vérifier les versions disponibles
gestvenv pyversions

# Installer la version Python manquante, puis réessayer
# Exemple: installer Python 3.9 puis
gestvenv create mon_env --python python3.9
```

Réinitialisation de la configuration

Problème: Configuration corrompue ou problèmes avec les environnements.

Solution:

```
# Supprimer le fichier de configuration
# Windows: %APPDATA%\GestVenv\config.json
# macOS: ~/Library/Application Support/GestVenv/config.json
# Linux: ~/.config/gestvenv/config.json

# Réinitialiser en exécutant une commande simple
gestvenv list
```

Pour toute question supplémentaire ou assistance, veuillez consulter la documentation en ligne ou ouvrir une issue sur le dépôt GitHub du projet.

```
# Afficher la documentation complète
gestvenv docs
```