

Scrippy : en finir avec script_final_old_v3_OK_last

Posté par Doug Le Tough (site web personnel) le 09 octobre 2026 à 20:12. Édité par Xavier Teyssier. Modéré par Julien Jorge. Licence CC By-SA.

Étiquettes : python, administration, système, script, devops

Tout administrateur système a croisé un jour, au détour d'un serveur, un `script_final_old_v3_OK_last.py` sans logs, sans historique, avec un mot de passe en clair à la ligne 42 et un commentaire `# TODO: gérer les erreurs` daté de 2014.

Après cinq ans de développement en parallèle de mes activités professionnelles, je publie **Scrippy**, un framework Python sous licence MIT pour écrire des scripts d'exploitation standardisés, robustes et prêts pour la production.

Sommaire

- [Le constat : le script d'exploitation, ce grand oublié](#)
 - [Le principe : un en-tête déclaratif](#)
 - [Un exemple vaut mieux qu'un long discours](#)
 - [Le double bénéfice](#)
 - [Les modules](#)
 - [Pas un prototype](#)
 - [Pourquoi seulement maintenant ?](#)
 - [La question qui fâche : et l'IA dans tout ça ?](#)
 - [Licence](#)
 - [Vos retours sont les bienvenus](#)

Le constat : le script d'exploitation, ce grand oublié

Dans beaucoup d'équipes, les scripts d'exploitation sont écrits vite, chacun à sa manière, puis maintenus dans la douleur. Le cycle de vie typique est bien connu :

- « C'est juste un petit script, je l'écris vite fait. »
- Le petit script part en production.
- L'auteur part en vacances (ou change d'entreprise).
- Le script plante un dimanche à 3 h du matin.
- L'astreinte découvre qu'il n'y a ni logs, ni historique, ni documentation, et que la configuration est codée en dur entre deux `print("ici")`.

Le tout agrémenté d'un mot de passe qui s'affiche fièrement dans la sortie standard. Oui, `hunter2`, on te voit.

Le principe : un en-tête déclaratif

Avec Scrippy, un script décrit son comportement dans un en-tête déclaratif : auteur, version, description, configuration attendue, options acceptées, utilisateurs autorisés, nombre d'exécutions simultanées. Une seule instruction `with` dans le code suffit ensuite pour bénéficier de l'ensemble des fonctionnalités, sans écrire une seule ligne de plomberie :

- un fichier de log par exécution, avec sortie colorée sur le terminal ;
- un historique complet des exécutions (paramètres, durée, code retour) ;
- un fichier de configuration par script, chargé et validé au démarrage ;
- des options de ligne de commande analysées et validées depuis l'en-tête, avec un `--help` généré automatiquement (personne n'aura plus d'excuse pour ne pas RTFM) ;
- le masquage automatique des secrets dans les logs et l'historique ;
- la restriction des utilisateurs et groupes autorisés à lancer le script ;
- une limite d'exécutions simultanées, avec file d'attente ;
- un répertoire de travail temporaire par exécution, supprimé automatiquement ;
- une gestion d'erreurs propre et des codes retour exploitables par votre ordonnanceur préféré.

Un exemple vaut mieux qu'un long discours

Voici un script de surveillance de l'occupation d'un système de fichiers. Toute ressemblance de l'auteur avec un personnage des Monty Python n'est évidemment pas fortuite : on fait du Python, on a des traditions.

```
#!/usr/bin/env python3
"""
-----
@author      : Luigi Vercotti
@date       : 2026-10-08
@version    : 1.0.0
@description : Monitor filesystem usage
-----
Configuration:
@conf:thresholds|warning|int|false
-----
Options:
@args:mount-point|str|Mount point to check
-----
"""
import shutil
import scrippy_core
from scrippy_core import logger

def main():
    with scrippy_core.ScriptContext() as _context
        usage = shutil.disk_usage(_context.args.mc
percent = round(usage.used * 100 / usage.t
        if percent >= _context.config.get("threshc
            logger.warning(f"[+] {_context.args.mour

if __name__ == '__main__':
    main()
```

Ce court script dispose déjà de `--help`, `--version`, d'un fichier de configuration validé, d'une option `--mount-point` validée, d'un log par exécution (`--log`), d'un historique (`--hist`), d'un mode debug (`--debug`) et d'un espace de travail temporaire. Le tout sans avoir vendu son âme à un fichier YAML de 800 lignes, ni avoir à relire une n-ième fois la doc de `argparse`, ni sans incantation mystique pour la configuration de log, ni pentacle renversé. Ni !

Le double bénéfice

D'une part, tous les scripts d'exploitation suivent le même standard : ils se lisent, s'exploitent et se maintiennent de la même façon, quel que soit leur auteur. Le script du collègue parti n'est plus une boîte noire.

D'autre part, ils s'écrivent beaucoup plus vite : inutile de recoder ces contrôles dans chaque script, on se concentre sur ce qu'il doit réellement faire. Le temps gagné peut être réinvesti dans des activités à haute valeur ajoutée, comme le troll sur systemd.

Les modules

Le cœur (`scrippy-core`) est complété par des modules optionnels :

- `scrippy-remote` : exécution distante et transferts de fichiers (SSH/SFTP, FTP/FTPS, CIFS) ;
- `scrippy-api` : client d'API REST, avec prise en charge d'OpenAPI ;
- `scrippy-db` : requêtes PostgreSQL, MySQL et SQLite ;
- `scrippy-mail` : envoi et lecture de courriels (SMTP, IMAP, POP3) ;
- `scrippy-snmp` : opérations SNMP `get`, `walk` et `set` ;
- `scrippy-git` : gestion de dépôts Git ;
- `scrippy-template` : génération de documents à partir de modèles Jinja2.

Pas un prototype

Scrippy n'est pas né d'un week-end pluvieux. Il tourne depuis plusieurs années en production, sur des usages bien réels. Il a donc survécu à des mises à jour, des migrations et des vendredis après-midi, ce qui constitue une forme de certification assez exigeante.

Pourquoi seulement maintenant ?

Parce que je n'étais pas satisfait de la documentation. Un outil mal documenté ou dont la documentation n'est pas à jour est un outil qu'on n'adopte pas, et, même si jusque là cela a été le cas pour Scrippy, « *Use The Source, Luke* » n'est pas une stratégie d'adoption. Je ne voulais pas publier un projet que d'autres ne pourraient pas s'approprier.

Dans les faits le code a toujours été publié mais n'ayant jamais fait aucune annonce l'adoption de Scrippy est relativement comparable à celle de GNU Hurd !

La question qui fâche : et l'IA dans tout ça ?

Avant que le Jihad butlérien ne se lève dans les commentaires, par souci de transparence : **le code de Scrippy a été écrit par un humain**, sur cinq ans.

Claude m'a aidé à remettre à plat la documentation, à l'enrichir d'un tutoriel progressif (voir [la documentation pas à pas](#)) et à maintenir à jour les README.md des dépôts. Il m'a également permis de détecter et corriger quelques bogues passés sous les radars malgré des années d'exploitation, ce qui est à la fois utile et clairement vexant.

L'IA reste un outil. Le meilleur des outils, entre les mains de quelqu'un qui ne sait pas ce qu'il fait, ne le rendra pas meilleur : un `im -rf` reste un `im -rf`, quelle que soit la qualité du clavier. C'est l'expérience, la compréhension du besoin et l'exigence qui font la qualité d'un projet. L'outil permet d'aller plus loin et plus vite.

Licence

Scrippy est un logiciel libre distribué sous licence MIT : vous pouvez l'utiliser, le modifier, le redistribuer et l'intégrer à vos projets, y compris commerciaux, à la seule condition de conserver la mention de copyright et le texte de la licence.

Vos retours sont les bienvenus

Si vous écrivez des scripts d'exploitation en Python, je serais ravi d'avoir vos retours, critiques comprises : la syntaxe de l'en-tête, les fonctionnalités manquantes, ou la comparaison avec ce que vous utilisez aujourd'hui.

Les rapports de bogues sont acceptés. Les trolls aussi, tant qu'ils sont de qualité (et vous pouvez y aller car mon Prumpleffer a été calibré par *t7h*).

Aller plus loin

-  [Le site officiel](#) (72 clics)
-  [La documentation pas à pas](#) (22 clics)
-  [Le code source](#) (30 clics)

(11 commentaires).

Markdown

EPUB

Pourquoi

Posté par Pouetpouet le 09 octobre 2026 à 21:33. Évalué à 0 (+1/-1).

Dans beaucoup d'équipes,

J'imagine que ça doit encore se faire, j'ai vécu ça aussi... mais créer un outils dédié pour gérer une mauvaise pratique me semble pas la bonne solution. Gérer sa production avec un outils de config management et le tout tracké dans un CVS c'est quand même la base d'une équipe un peu sérieuse

Répondre

[^] # Re: Pourquoi

Posté par Doug Le Tough (site web personnel) le 09 octobre 2026 à 23:00. Évalué à 4 (+3/-0).

Il est évidemment recommandé que ces scripts soient centralisés, versionnés et déployés de manière automatisée. L'un n'empêche pas l'autre, pas plus qu'un CVS et un outil de gestion de configuration n'empêchent `script_final_old_v3_OK_last.py` d'être déployé en prod (déjà vu).

L'idée est d'apporter plusieurs choses et pour lister les principales:

- Un cadre aux scripts d'exploitation, de manière à gagner en temps de développement et de maintenance (gestion déclarative des paramètres de configuration, des options, configuration du logger, etc.). Cette étape représente généralement la plus grande partie du développement d'un script d'exploitation, alors que c'est peu ou prou toujours la même chose (i.e. valider la configuration du script, jouer avec les arcanes d'argparse pour gérer les options, configurer un logger, gérer les informations sensibles dans les sorties).
- Un historique d'exécution des scripts qui indique l'heure de début, l'heure de fin, la durée d'exécution, l'utilisateur qui a exécuté le script, si le script a été exécuté via `sudo` (et, le cas échéant, qui a exécuté le `sudo`), le code retour du script, et, si le script s'est terminé sur une erreur, quelle a été cette erreur, etc. Et je parle d'un historique configurable : par défaut, tu peux consulter les 50 dernières exécutions, mais tu peux décider de conserver les 200 dernières. Le genre de truc qui permet de vérifier rapidement pourquoi le script `mon_script.py` a planté il y a 15 jours sans que personne ne s'en aperçoive.
- Des logs standardisés, la possibilité de gérer les exécutions parallèles, d'éviter la fuite d'informations sensibles dans les logs, etc., mais également de consulter les logs des dernières exécutions. Parce que lorsque le script critique, exécuté par un ordonnanceur toutes les 5 minutes, a planté un dimanche à 3 heures du matin, tu veux savoir pourquoi, même s'il t'a fallu 30 minutes pour te réveiller. Et dans ces moments-là, tu ne veux pas avoir à réfléchir (où est le log ? Comment y accéder ? Comment identifier le bon log parmi les gazillions de logs conservés ?). Avec Scrippy, c'est simplement `mon_script.py --hist` pour visualiser l'historique d'exécution et identifier l'exécution qui a planté, puis `mon_script.py --log <SESSION_ID>` pour visualiser le log qui contient la raison du plantage.
- Bien d'autres choses choses que je vous laisse découvrir grâce au [tutoriel](#) :-)

Évidemment, Scrippy n'est pas magique. Comme indiqué dans la dépêche à propos de l'IA, le meilleur outil donné à un mauvais ouvrier n'en fera pas un bon ouvrier.

Répondre

Tu n'as pas compris

Posté par cg le 09 octobre 2026 à 23:05. Évalué à 8 (+6/-0).

