Explication détaillée, ligne par ligne, d'un script Bash de recherche

Script: `search_mtime`, `find_suspects`, `search_for_type`

Contents

1 Script complet (avec numéros de ligne)

```
1  #!/usr/bin/env bash
2  set -Eeuo pipefail
3  IFS=$'\n\t'
4
5  WORKING_DIR="$(cd -- "$(dirname -- "${BASH_SOURCE[0]}")" && pwd -P)"
6
7  # shellcheck source=./constants.sh
8  if [[ -f "$WORKING_DIR/constants.sh" ]]; then
9    # shellcheck disable=SC1091
10   source "$WORKING_DIR/constants.sh"
11 else
12   CYAN_COLOR=""
13   RED_COLOR=""
14   NO_COLOR=""
15 fi
16
17 search_mtime() {
18     printf "%s searching files edited within 24h : %s\n" "${CYAN_COLOR:-}" "${NO_COLOR
       :-}"
19     find . -type f -mtime -1 -print
20 }
21
22 find_suspects() {
23     local ext=${1:-}
24     if [[ -z "$ext" ]]; then
25         printf "Usage: %s <extension>\n" "${FUNCNAME[0]}"
26         return 2
27     fi
28     printf "%s searching suspect files with .%s extension : %s\n" "${RED_COLOR:-}" "
       $ext" "${NO_COLOR:-}"
29     find . -type f -iname "*.${ext}" -print0 \
30       | xargs -0 -r grep -HnEI -- 'malware|virus|worm' || true
31 }
32
33 search_for_type() {
34     local ext=${1:-}
35     if [[ -z "$ext" ]]; then
36         printf "Usage: %s <extension>\n" "${FUNCNAME[0]}"
37         return 2
38     fi
39     printf "search for type => %s :\n" "$ext"
```

```
40      find . -type f -iname "*.${ext}"
41  }
42
43  # case "${1:-}" in
44  #   mtime)        search_mtime ;;
45  #   suspects)     shift; find_suspects "${1:-}" ;;
46  #   type)         shift; search_for_type "${1:-}" ;;
47  #   *)            printf "Usage: %s {mtime|suspects <ext>|type <ext>}\n" "$0"; exit 1
        ;;
48  # esac
```

2 Explication ligne par ligne

Les références aux numéros ($L\#$) renvoient aux lignes listées en section **??**.

En-tête et garde-fous du shell : L1–L3

L1 `#! /usr/bin/env bash` : Shebang. Demande à l'OS d'exécuter ce fichier avec `bash` trouvé via `env`. Plus portable que `#!/bin/bash`.

L2 `set -Eeuo pipefail` : Active des protections essentielles :

- `-E` : propage les traps `ERR` dans les fonctions/sous-shells.
- `-e` : quitte si une commande renvoie un code non nul (sauf cas contrôlés).
- `-u` : erreur si une variable non définie est référencée.
- `pipefail` : un pipeline échoue si *n'importe quel* maillon échoue (et pas seulement le dernier).

L3 `IFS=$'\n\t'` : redéfinit le séparateur de champs interne au retour-ligne et tabulation. Évite les découpes hasardeuses sur les espaces dans les chemins.

Résolution fiable du dossier du script : L5

L5 `WORKING_DIR="...$ {BASH_SOURCE[0]}..."` :

- `${BASH_SOURCE[0]}` : chemin du *fichier* courant (plus robuste que `$0` si le script est *sourcé*).
- `dirname - "..."` : isole le dossier parent. Le `-` protège si le chemin commence par `-`.
- `cd - "..." && pwd -P` : se place dans ce dossier puis affiche le chemin *physique* (résolution des liens symboliques) et l'enregistre dans `WORKING_DIR`.

Chargement optionnel des constantes : L7–L15

L7 Commentaire `shellcheck source=./constants.sh` : aide l'outil *ShellCheck* à localiser le fichier sourcé.

L8 `if [[ -f "$WORKING_DIR/constants.sh" ]]` : on charge le fichier de constantes s'il existe dans le même dossier.

L9 `# shellcheck disable=SC1091` : désactive un avertissement (fichier non détectable statiquement).

L10 `source "$WORKING_DIR/constants.sh"` : importe les variables/fonctions (typiquement, couleurs ANSI).

L11–L15 Sinon, définit des *fallbacks* vides pour `CYAN_COLOR`, `RED_COLOR`, `NO_COLOR` afin d'éviter toute référence de variable non définie (`set -u`).

`search_mtime` : L17–L20

L17 Déclaration de la fonction.

L18 `printf "%s ... %s\n" "${CYAN_COLOR:-}" "${NO_COLOR:-}"` :

- `printf` est plus prévisible que `echo -e`.
- `${VAR:-}` injecte une valeur vide si la variable n'est pas définie.

L19 `find . -type f -mtime -1 -print` :

- `-type f` : fichiers uniquement.
- `-mtime -1` : modifiés dans les dernières 24 heures.
- `-print` : affiche les chemins.

L20 Fin de la fonction.

`find_suspects` : L22–L31

L22 Déclaration de la fonction.

L23 `local ext=${1:-}` : récupère l'extension passée en premier argument (vide si absent).

L24–L27 Si l'argument est vide, affiche un *usage* et retourne le code 2 (mauvais usage).

L28 En-tête coloré *(si couleurs disponibles)* avec l'extension ciblée.

L29–L30 Recherche et filtrage :

- `find . -type f -iname "*.${ext}" -print0` : liste les fichiers de l'extension (casse insensible), en séparant les résultats par NUL (`-print0`) pour être sûr même si les chemins contiennent espaces/retours-ligne.
- `| xargs -0 -r grep -HnEI - 'malware|virus|worm'` :
  - `-0` : lit des chemins séparés par NUL.
  - `-r` : n'exécute rien si l'entrée est vide (évite `grep` sans argument).
  - `grep -HnEI` :
    - `-H` : affiche toujours le nom de fichier.
    - `-n` : affiche le numéro de ligne.
    - `-E` : expressions régulières étendues (pour le `|` alternation).
    - `-I` : ignore les fichiers binaires.
  - motif `'malware|virus|worm'` : cherche l'un de ces termes.
- `|| true` : en cas d'absence de correspondance, `grep` renvoie 1. Cette clause empêche `set -e` d'arrêter le script.

L31 Fin de la fonction.

`search_for_type` : L33–L41

L33 Déclaration de la fonction.

L34 `local ext=${1:-}` : extension visée.

L35–L38 Validation de l'argument : usage sinon, retour code 2.

L39 En-tête d'information.

L40 `find . -type f -iname "*.${ext}"` : liste tous les fichiers de cette extension, sans tenir compte de la casse.

L41 Fin de la fonction.

Dispatcher optionnel (commenté) : L43–L48

L43–L48 Bloc `case` montrant comment transformer le fichier en *petit utilitaire* :

- `mtime` → `search_mtime`
- `suspects <ext>` → `find_suspects <ext>`
- `type <ext>` → `search_for_type <ext>`
- sinon, message d'usage et sortie avec code 1.

Les `shift` retirent le premier argument (la sous-commande) pour que l'argument suivant soit l'extension.

3 Notes de sûreté et de portabilité

- **Pourquoi `BASH_SOURCE` plutôt que `$0` ?** `BASH_SOURCE` donne le *fichier* réellement sourcé, fiable dans les fonctions et lors d'un `source`. `$0` peut pointer vers le shell ou un wrapper.
- **Pourquoi `printf` ?** `echo -e` a des divergences d'implémentation selon les shells; `printf` est normalisé POSIX.
- **Pourquoi `-print0/-0` ?** Sécurité face aux espaces, accents et retours ligne dans les chemins (sinon `xargs` sépare sur espaces).
- **Pourquoi `|| true` après `grep` ?** Avec `set -e`, un `grep` sans match retourne 1 et tuerait le script; ici on considère "aucun match" comme un cas non fatal.
- **Casse insensible dans `grep`** : ajoutez `-i` à `grep` si vous souhaitez ignorer la casse pour le texte (`-I` ne concerne que les binaires).

4 Variantes utiles

- **Plusieurs extensions d'un coup** : remplacer la recherche par un motif étendu, ex. `-iregex '.*\.(php|js|ts)'` ou itérer sur une liste d'extensions.
- **Rapport JSON/CSV** : canaliser la sortie `grep` vers un formateur (awk/python) pour produire un inventaire machine-readable.
- **Exclusion de dossiers** : ajouter `-not -path "./vendor/*"` ou `-prune` pour ignorer `.git`, `node_modules`, etc.

5 Exemples d'usage

- **Fichiers modifiés <24h** : appeler la fonction `search_mtime`.
- **Termes suspects dans les `.php`** : `find_suspects php`.
- **Lister tous les `.js`** : `search_for_type js`.
- **Avec le dispatcher (décommenté)** :
  - `./script.sh mtime`
  - `./script.sh suspects js`
  - `./script.sh type log`