Tutorial: Automating File Organization with a Bash Script

In this tutorial, we'll walk through a bash script that organizes files in a directory based on their extensions. This script also moves certain folders to a new location, excludes others entirely, and changes file permissions after they've been organized.

The script is designed for automation, efficiency, and clarity. I'll explain each part and why it's structured this way, keeping things practical and straightforward.

Step 1: Define the Environment

The first thing we do is define the source and target directories. You'll need to customize these paths to fit your setup.

```
#!/bin/bash

SOURCE_DIR="/path/to/source"
TARGET_DIR="/path/to/target"
USER="paulgrey"
```

- `SOURCE_DIR`: The directory where all the files are located.
- `TARGET_DIR`: The directory where you want to organize and move the files.
- `USER`: The system user for setting file ownership. This is important to ensure proper permissions are applied after moving files.

Why this structure?

We set up clear, readable variable names to define important paths. This makes the script portable and easy to modify later. It's easier to adjust a path in one place than in multiple lines of code.

Step 2: Handling Folder Exclusion

We have two arrays for excluding and ignoring certain folders:

```
EXCLUDE_FOLDERS=("Backup" "paulgrey" "timeshift")
IGNORE_FOLDERS=(".lost+found" ".Trash-1000" "lost+found" "Trash-1000")
```

- **EXCLUDE_FOLDERS**: Folders that we want to move as is, without reorganizing their contents.
- **IGNORE_FOLDERS**: Folders that we want to completely ignore and skip over.

Why exclude or ignore folders?

You often have folders that need special handling or that shouldn't be touched. For example, backup folders or trash folders contain files you don't want to organize. This section ensures they are either moved in their entirety or ignored during the process.

Step 3: Define File Types and Extensions

Next, we create a `declare -A extensions` statement to define an associative array that links directories to specific file extensions:

```
declare -A extensions
extensions+=( ["Images/PNG"]="png" ["Images/JPG"]="jpg" ... )
```

This part is divided into categories like Images, Documents, Executables, and so on, to handle different file types. Each category has a folder and a list of corresponding file extensions.

Why use associative arrays?

This method is efficient because it allows us to pair a folder with multiple file extensions. It keeps the logic simple when we later use this data to move files based on their type.

Step 4: Creating Target Directories

Now we need to create the directories where the organized files will be stored:

```
create_directories() {
    for ext in "${!extensions[@]}"; do
        mkdir -p "$TARGET_DIR/${ext%/*}/${ext##*/}"
    done
}
```

- `mkdir -p`: This ensures that the directory will be created only if it doesn't already exist.
- `${ext%/*}` and `${ext##*/}`: These extract the main folder name (like `Images`) and the subfolder (like `PNG`) to create the correct structure.

Why this function?

Before moving files, we need to ensure the target directory structure exists. This function ensures all necessary directories are created beforehand.

Step 5: Moving Files Based on Extension

We then define a function to move files to their correct directories based on their extensions:

```
move_files() {
    for ext in "${!extensions[@]}"; do
        find "$SOURCE_DIR" -type f -iname "*.${extensions[$ext]}" |
        while read -r file; do
            if is_ignored_folder "$file"; then
                continue
            fi
            mv "$file" "$TARGET_DIR/${ext%/*}/${ext##*/}"
            chown "$USER:$USER"
        "$TARGET_DIR/${ext%/*}/${ext##*/}/$(basename "$file")"
            chmod 755 "$TARGET_DIR/${ext%/*}/${ext##*/}/$(basename
        "$file")"
        done
    done
}
```

- `find "$SOURCE_DIR" -type f -iname "*.${extensions[$ext]}"`: This finds all files matching the current extension.
- `mv "$file"`: Moves the file to its corresponding folder.
- `chown "$USER:$USER"` and `chmod 755`: Changes the ownership and permissions of the file after moving.

Why use `find`?

`find` is a powerful tool for searching files. It allows us to search recursively within the source directory and filter out only the files with specific extensions. It's perfect for handling large directories where files may be buried deep in subfolders.

Step 6: Handling Excluded Folders

We also need a function to handle the excluded folders (those we want to move without organizing):

```
move_excluded_folders() {
    for folder in "${EXCLUDE_FOLDERS[@]}"; do
        if [ -d "$SOURCE_DIR/$folder" ]; then
            mv "$SOURCE_DIR/$folder" "$TARGET_DIR/"
            chown -R "$USER:$USER" "$TARGET_DIR/$folder"
        fi
    done
}
```

Why move these folders directly?

Some folders may contain files that don't need to be individually organized, like backup directories. This function ensures that entire folders can be moved and owned correctly without reorganizing their contents.

Step 7: Ignoring Specific Folders

We also need to check whether a file is inside an ignored folder. For this, we use:

```
is_ignored_folder() {
    local folder="$1"
    for ignore in "${IGNORE_FOLDERS[@]}"; do
        if [[ "$folder" == *"$ignore"* ]]; then
            return 0
        fi
    done
    return 1
}
```

Why use a check for ignored folders?

Certain system folders (like `.Trash` or `lost+found`) should be completely skipped. This function ensures those folders are ignored during the entire process, preventing unnecessary operations.

Step 8: Putting it All Together

At the end of the script, we combine everything:

```
create_directories
move_excluded_folders
move_files

echo "Files have been organized, excluded folders have been moved, and
        permissions have been set."
```

Why this order?

- **Create directories** first so that when we move files, we know the target folders exist.
- **Move excluded folders** before organizing files to ensure nothing is mixed with the folders we want untouched.
- **Move files** last, since this is the heaviest operation.

Final Thoughts

This script provides an efficient, automated way to organize files based on their extensions, while carefully handling special cases for excluded and ignored folders. It's flexible and can be adapted by modifying the source and target directories or adding new file extensions to the list.

Feel free to extend this script further based on your specific needs, whether you want to add new file types or modify folder exclusion rules.