Hyprfiles

« Configuration d'un environnement de bureau Arch Linux utilisant le compositeur Wayland Hyprland. »

Par Vasco – En 2024 – Projet Personnel

Table des matières

1. Introduction

Ce projet consistait en la configuration d'un environnement de bureau basé sur Arch Linux et utilisant Hyprland comme compositeur Wayland. Il intègre une suite de fichiers de configuration (ou "dotfiles"), des scripts Bash et Python sur mesure pour automatiser des tâches, gérer les paramètres système (volume, luminosité, ventilateurs, WiFi), et offrir une expérience utilisateur cohérente et esthétique avec le thème Catppuccin Mocha. L'objectif était de créer un environnement de travail optimisé, performant et visuellement agréable.

2. Objectifs et Apprentissages du Projet

2.1 Contexte de Développement

Ce projet est né de ma volonté de créer un environnement de bureau Linux sur mesure. Je souhaitais en apprendre un peu plus sur le fonctionnement de ce système d'exploitation que j'utilise au travail. Pour ce faire je m'étais donné le défit de plonger tête la première dans cet univers en essayant de créer mon environnement de rêve, impossible sans l'utilisation d'un système libre tel que Linux.

2.2 Objectifs et Apprentissages Clés

- **Maîtrise de Linux (Usage Personnel et Professionnel) :** Approfondir l'utilisation quotidienne et avancée de Linux, notamment Arch Linux ;
- **Configuration d'Outils Linux Populaires :** Apprendre à configurer en détail des composants essentiels d'un environnement de bureau moderne (Hyprland, Waybar, Rofi, Neovim, Dunst) ;
- **Scripting pour l'Automatisation et l'Extension des Fonctionnalités :**
    - Utiliser des scripts Bash pour la gestion du système (contrôle du volume, de la luminosité, des médias Spotify) ;
    - Développer des scripts Python pour créer des interfaces utilisateur interactives avec Rofi (menus pour la gestion des ventilateurs, du WiFi, de l'alimentation) ;
    - Créer de nouvelles fonctionnalités comme des notifications personnalisées pour le volume, la luminosité, etc.
- **Compréhension Technique Approfondie de Linux :**
    - Explorer le fonctionnement interne du système à travers la configuration d'outils et la résolution de problèmes ;
    - **Gestion des Pilotes Graphiques Complexes :** Relever le défi de la configuration et de la correction des pilotes graphiques NVIDIA sur un ordinateur portable hybride (GPU NVIDIA discret + GPU Intel intégré).
- **Personnalisation Esthétique (ou "Ricing") :** Créer une interface utilisateur visuellement cohérente et agréable en utilisant le thème Catppuccin Mocha (que j'aime particulièrement) sur l'ensemble des applications et composants ;
- **Optimisation des Performances :** Rechercher des solutions performantes, notamment en envisageant la migration de scripts Python vers Bash ou C pour les menus Rofi ;

- **Gestion de Configuration (Dotfiles) :** Organiser et versionner les fichiers de configuration pour faciliter la maintenance, le partage et la reproductibilité de l'environnement.

3. Architecture de l'Environnement

3.1. Philosophie et Structure Générale

L'environnement est conçu autour de la légèreté, de la performance et de la personnalisation offertes par Arch Linux et le compositeur Wayland Hyprland. La philosophie était de construire un système minimaliste mais hautement fonctionnel, où chaque composant est choisi et configuré avec soin. L'automatisation via des scripts joue un rôle central pour étendre les fonctionnalités et simplifier les interactions.

3.2. Composants Logiciels Clés

- **Distribution :** Arch Linux ;
- **Compositeur Wayland :** Hyprland ;
- **Barre de Statut :** Waybar ;
- **Lanceur d'Applications / Menus :** Rofi (avec des modes personnalisés en Python) ;
- **Terminal :** Kitty ;
- **Gestionnaire de Fichiers :** Thunar ;
- **Navigateur Web :** Firefox ;
- **Éditeur de Texte Principal :** Neovim ;
- **Système de Notifications :** Dunst ;
- **Gestion Audio :** pamixer ;
- **Gestion Luminosité :** brightnessctl ;
- **Contrôle Média :** playerctl ;
- **Contrôle Ventilateurs :** nbfc-linux (via un script Rofi en Python) ;
- **Gestion Réseau :** nmcli (NetworkManager) ;
- **Verrouillage d'Écran :** hyprlock ;
- **Shell :** Zsh avec Oh My Zsh et plugins (zsh-syntax-highlighting, zsh-autosuggestions).

3.3. Structure des Fichiers de Configuration (dotfiles)

Les fichiers de configuration sont organisés de manière centralisée, principalement dans le répertoire ~/.config/, un dossier normalisé pour les fichiers de configuration sous linux.

4. Configuration des Composants Principaux

4.1. Hyprland (Compositeur Wayland)

- **Fichier Principal :** ~/.config/hypr/hyprland.conf ;

- **Configuration Clé :**
    - **Multi-Moniteurs :** Définition et agencement de plusieurs écrans avec des résolutions et taux de rafraîchissement spécifiques (ex: $mainMonitor = eDP-1, $leftMonitor, $middleMonitor). Inclut la transformation d'un écran en mode portrait ;
    - **Applications par Défaut :** Assignation de variables pour le terminal ($terminal = kitty), gestionnaire de fichiers ($fileManager = Thunar), navigateur ($browser = firefox), et menu ($menu = rofi -show drun) ;
    - **Esthétique Visuelle :** Bordures arrondies (rounding = 10), gestion de la transparence active/inactive (opacité de 0.95), animations fluides avec des courbes de Bézier personnalisées ;
    - **Thème :** Intégration du thème de couleurs Catppuccin Mocha (couleur principale rgb(cba6f7)) ;
    - **Raccourcis Clavier :** Gérés dans un fichier séparé keybinds.conf pour une meilleure organisation ;
    - **Espaces de Travail :** Définis dans workspaces.conf, supportant jusqu'à 30 espaces de travail avec une navigation via des modificateurs multiples.

4.2. Waybar (Barre de statut)

- **Configuration :** Fichiers JSONC (config) et CSS (style.css) dans ~/.config/waybar/ ;
- **Architecture Modulaire :** La barre est divisée en sections (modules-left, modules-center, modules-right) ;
    - **Modules Standards :** hyprland/workspaces, network, battery, backlight, pulseaudio, clock, temperature, cpu, memory, power-profiles-daemon ;
    - **Modules Personnalisés (custom/) :**
        - custom/appmenu : Lanceur d'applications Rofi ;
        - custom/fan : Affiche l'état actuel des ventilateurs et permet d'ouvrir le fanmenu Rofi ;
        - custom/power : Raccourci vers le powermenu Rofi ;
- **Stylisation :** CSS personnalisé pour s'intégrer au thème Catppuccin Mocha.

4.3. Rofi (Lanceur d'applications et menus interactifs)

- **Configuration :** Fichiers .rasi pour le thème de chaque menu, situés dans les sous-dossiers de ~/.config/rofi/modes/ ;
- **Modes Personnalisés (Scripts Python) :**
    - fanmenu : Interface pour contrôler les modes de ventilation via nbfc-linux ;
    - powermenu : Menu simple pour éteindre, redémarrer, verrouiller ou déconnecter ;
    - wifimenu : Gestionnaire WiFi complet interagissant avec nmcli.

- **Thème :** Cohérent avec Catppuccin, utilisant des variables CSS dans les fichiers .rasi.

4.4. Neovim (Éditeur de texte)

- **Configuration :** Principalement en Lua, dans ~/.config/nvim/ ;
- **Structure de Plugins (gérée par un gestionnaire comme Lazy.nvim, bien que non explicitement nommé) :**
    - **LSP (Language Server Protocol) :** Configuration avec nvim-lspconfig et mason.nvim pour l'installation automatique des serveurs de langage ;
    - **Auto-Complétion :** nvim-cmp avec de multiples sources (LSP, snippets, buffers) ;
    - **Interface Utilisateur :** Thème TokyoNight (qui se marie bien avec Catppuccin), barre de statut Lualine, écran d'accueil Alpha ;
    - **Productivité :** Telescope (fuzzy finder), Nvim-Tree (explorateur de fichiers), Comment.nvim, Nvim-surround.

4.5. Autres Outils

- **Kitty (Terminal) :** Configuré pour le thème Catppuccin, la police FiraCode Nerd Font ;
- **Thunar (Gestionnaire de Fichiers) :** Intégration du thème GTK Catppuccin ;
- **Zsh et Oh My Zsh :** Configuration du prompt avec Oh My Posh (utilisant un thème Catppuccin), et plugins pour l'auto-suggestion et la coloration syntaxique.

5. Scripts Personnalisés et Automatisation

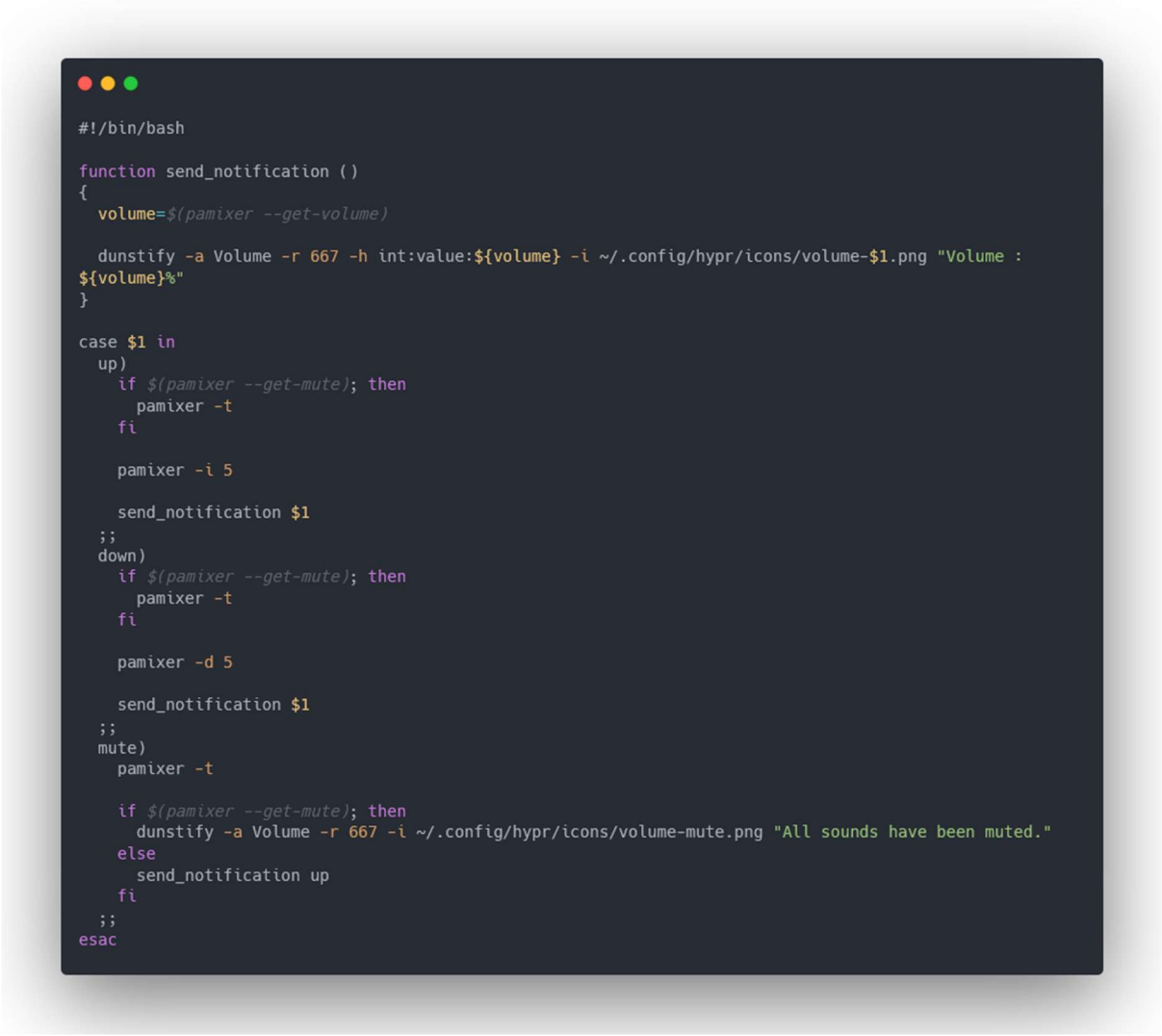
5.1. Vue d'ensemble des Scripts

Le projet s'appuie fortement sur des scripts pour étendre les fonctionnalités de base de l'environnement. Ils sont divisés en scripts Bash pour les interactions système directes et rapides, et en scripts Python pour les interfaces utilisateur plus complexes via Rofi.

5.2. Scripts Bash (Contrôles système et multimédia)

Situés dans ~/.config/hypr/scripts/.

- **volume.sh :**
    - **Fonctionnalité :** Augmente, diminue ou coupe le volume audio principal ;
    - **Logique :** Utilise pamixer. Envoie une notification dunstify avec une icône et une barre de progression indiquant le niveau de volume actuel.



```bash
#!/bin/bash

function send_notification ()
{
  volume=$(pamixer --get-volume)

  dunstify -a Volume -r 667 -h int:value:${volume} -i ~/.config/hypr/icons/volume-$1.png "Volume :
${volume}%"
}

case $1 in
  up)
    if $(pamixer --get-mute); then
      pamixer -t
    fi

    pamixer -i 5

    send_notification $1
  ;;
  down)
    if $(pamixer --get-mute); then
      pamixer -t
    fi

    pamixer -d 5

    send_notification $1
  ;;
  mute)
    pamixer -t

    if $(pamixer --get-mute); then
      dunstify -a Volume -r 667 -i ~/.config/hypr/icons/volume-mute.png "All sounds have been muted."
    else
      send_notification up
    fi
  ;;
esac
```

- **microphone.sh :**
    - **Fonctionnalité :** Contrôle le volume et l'état muet du microphone par défaut ;
    - **Logique :** Utilise pamixer --default-source. Désactive automatiquement le mode muet lors de l'augmentation du volume. Notifications dunstify distinctes.

(logique similaire à celle du script volume.sh)

- **brightness.sh :**
    - **Fonctionnalité :** Ajuste la luminosité de l'écran ;
    - **Logique :** Utilise brightnessctl. Calcule le pourcentage de luminosité (basé sur une valeur maximale spécifique, ex: 960) et l'affiche via dunstify.

```
#!/bin/bash

function send_notification ()
{
  brightness=$(($(brightnessctl g)/960))

  dunstify -a Brightness -r 96000 -h int:value:${brightness} -i ~/.config/hypr/icons/brightness-$1.png
"Screen Brightness : ${brightness}%"
}

case $1 in
  up)
    brightnessctl s 5%+
  ;;
  down)
    brightnessctl s 5%-
  ;;
esac

send_notification $1
```

- **spotify-control.sh :**
    - **Fonctionnalité :** Contrôle la lecture de Spotify (précédent, suivant, lecture/pause) ;
    - **Logique :** Utilise playerctl --player=spotify. Vérifie d'abord si Spotify est en cours d'exécution. Pour la lecture/pause, récupère les métadonnées (titre, artiste), télécharge la pochette de l'album avec curl, et affiche une notification dunstify enrichie avec la pochette et une barre de progression.

```bash
#!/bin/bash

players=$(playerctl -l)

if ! echo "$players" | grep -q "spotify"; then
  exit 1
fi

function get_time ()
{
  duration=$(playerctl --player=spotify metadata --format "{{duration(mpris:length - position)}}")
  IFS=':' read -r minutes seconds <<< "$duration"
  duration=$(expr $minutes \* 60 + $seconds)

  position=$(playerctl --player=spotify metadata --format "{{duration(position)}}")
  IFS=':' read -r minutes seconds <<< "$position"
  position=$(expr $minutes \* 60 + $seconds)

  percentage=$((100 * $position / $duration))

  echo $percentage
}

case $1 in
  previous)
    playerctl --player=spotify previous
  ;;
  next)
    playerctl --player=spotify next
  ;;
  pause)
    playerctl --player=spotify play-pause

    info=$(playerctl --player=spotify metadata --format "{{title}} - {{artist}}")
    time=$(playerctl --player=spotify metadata --format "--- {{duration(position)}} /
{{duration(mpris:length)}} ---")
    time_value=$(get_time)
    image=$(playerctl --player=spotify metadata --format "{{mpris:artUrl}}")
    curl $image --output /tmp/spotify-img.png --silent

    state=$(playerctl --player=spotify status)

    dunstify -a Music -r 555 -h int:value:$time_value -i /tmp/spotify-img.png "⏯ $state music."
"$info<br>$time" -t 2000
  ;;
esac
```

- **spotify-info.sh :**
    - **Fonctionnalité :** Affiche les informations de la piste Spotify en cours de lecture (pour hyprlock) ;
    - **Logique :** Utilise playerctl pour récupérer le statut et les métadonnées. Formate la sortie avec des icônes de statut (lecture/pause).

```
#!/bin/bash

case $1 in
  text)
    status=$(playerctl --player=spotify status)
    info=$(playerctl --player=spotify metadata --format "{{title}} - {{artist}}")

    case $status in
      Playing)
        echo "▶ $info"
      ;;
      Paused)
        echo "⏸ $info"
      ;;
    esac
  ;;
  time)
    time=$(playerctl --player=spotify metadata --format "--- {{duration(mpris:length - position)}}
---")

    echo "$time"
  ;;
esac
```

5.3. Scripts Python pour Rofi (Menus interactifs)

Situés dans ~/.config/rofi/modes/ (chacun dans son sous-dossier avec son script run.py et son fichier de configuration Rofi .rasi).

- **fanmenu/run.py (Contrôle des Ventilateurs) :**
    - **Fonctionnalité :** Offre une interface Rofi pour sélectionner différents modes de ventilation (Silencieux 0%, Auto, Faible 30%, Moyen 65%, Maximum 100%) ;
    - **Logique :** Interagit avec nbfc set -s [speed] ou nbfc set -a. Lit et écrit l'état actuel dans un fichier status.txt pour préselectionner l'option active lors de l'ouverture du menu. Utilise des icônes Unicode pour représenter les modes.

```python
import subprocess

# Customization
config_file = "$HOME/.config/rofi/modes/fanmenu/config.rasi"
status_file = "/home/svasco/.config/rofi/modes/fanmenu/status.txt"

options = "\n\n\n\n"

# Get the current status
current_status = ""
with open(status_file, "r") as status:
    current_status = status.readline().strip()

# Display the modes selector
selected = subprocess.run(
    f'echo -en "{options}" | rofi -config {config_file} -dmenu -select "{current_status}"',
    shell=True,
    stdout=subprocess.PIPE,
    stderr=subprocess.PIPE,
    text=True,
)

if selected.stderr:
    print(selected.stderr)
    quit()

selected = selected.stdout.strip()

match selected:
    case "":
        subprocess.run(f'nbfc set -s 0 | echo "{selected}" > {status_file}', shell=True)
    case "":
        subprocess.run(f'nbfc set -a | echo "{selected}" > {status_file}', shell=True)
    case "":
        subprocess.run(
            f'nbfc set -s 30 | echo "{selected}" > {status_file}', shell=True
        )
    case "":
        subprocess.run(
            f'nbfc set -s 65 | echo "{selected}" > {status_file}', shell=True
        )
    case "":
        subprocess.run(
            f'nbfc set -s 100 | echo "{selected}" > {status_file}', shell=True
        )
    case _:
        quit()
```

(Script à titre indicatif)

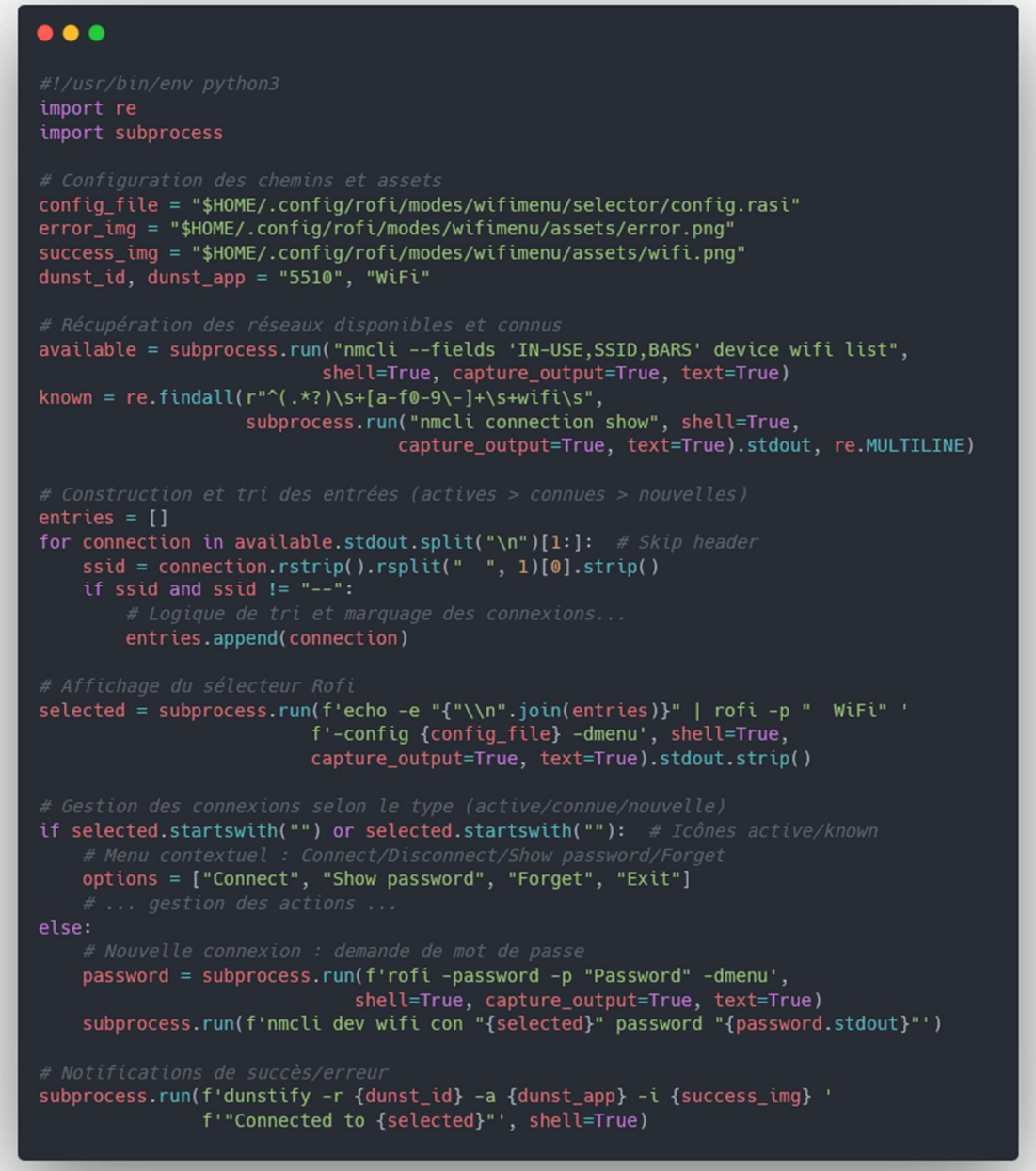
- **powermenu/run.py (Menu d'Alimentation) :**
    - **Fonctionnalité :** Propose un menu Rofi simple pour les actions d'alimentation : Arrêt, Redémarrage, Verrouillage de l'écran, Déconnexion ;
    - **Logique :** Exécute les commandes système correspondantes (poweroff, reboot, hyprlock) via subprocess.run(). Utilise des icônes Unicode.

```python
import subprocess

options = "\n\n\n"
config_file = "$HOME/.config/rofi/modes/powermenu/config.rasi"

selected = subprocess.run(
    f'echo -e "{options}" | rofi -config {config_file} -dmenu',
    shell=True,
    stdout=subprocess.PIPE,
    stderr=subprocess.PIPE,
    text=True,
)

if selected.stderr:
    print(selected.stderr)
    quit()

match selected.stdout.strip():
    case "":
        subprocess.run("poweroff", shell=True)
    case "":
        subprocess.run("reboot", shell=True)
    case "":
        subprocess.run("hyprlock", shell=True)
    case "":
        subprocess.run("logout", shell=True)
    case _:
        quit()
```

(Script à titre indicatif)

- **wifimenu/run.py (Gestionnaire WiFi) :**
    - **Fonctionnalité :** Un gestionnaire WiFi complet via Rofi, permettant de lister les réseaux disponibles et connus, de se connecter (avec saisie de mot de passe si nécessaire), de se déconnecter, d'afficher un mot de passe sauvegardé, et d'oublier un réseau ;
    - **Logique Complexe :**
        1. Récupère les réseaux WiFi disponibles et connus en utilisant nmcli device wifi list et nmcli connection show ;
        2. Traite et normalise les SSID (limite de 26 caractères, ajout d'ellipses si plus long) ;

3. Affiche une liste triée dans Rofi (réseau actif en premier, puis connus, puis nouveaux), avec des icônes distinctives ;
4. Si un réseau nécessite un mot de passe et qu'il n'est pas connu, un second menu Rofi (mode password) est affiché pour la saisie ;
5. Gère les tentatives de connexion avec nmcli dev wifi con "[SSID]" (password "[PASSWORD]") ;
6. Affiche des notifications dunstify pour les succès, échecs (mot de passe incorrect, réseau introuvable), et autres actions ;
7. Propose un menu contextuel pour les réseaux déjà connus ou actifs (déconnecter, afficher mot de passe, oublier).



```python
#!/usr/bin/env python3
import re
import subprocess

# Configuration des chemins et assets
config_file = "$HOME/.config/rofi/modes/wifimenu/selector/config.rasi"
error_img = "$HOME/.config/rofi/modes/wifimenu/assets/error.png"
success_img = "$HOME/.config/rofi/modes/wifimenu/assets/wifi.png"
dunst_id, dunst_app = "5510", "WiFi"

# Récupération des réseaux disponibles et connus
available = subprocess.run("nmcli --fields 'IN-USE,SSID,BARS' device wifi list",
                           shell=True, capture_output=True, text=True)
known = re.findall(r"^(.*?)\s+[a-f0-9\-]+\s+wifi\s",
                   subprocess.run("nmcli connection show", shell=True,
                                  capture_output=True, text=True).stdout, re.MULTILINE)

# Construction et tri des entrées (actives > connues > nouvelles)
entries = []
for connection in available.stdout.split("\n")[1:]:  # Skip header
    ssid = connection.rstrip().rsplit("  ", 1)[0].strip()
    if ssid and ssid != "--":
        # Logique de tri et marquage des connexions...
        entries.append(connection)

# Affichage du sélecteur Rofi
selected = subprocess.run(f'echo -e "{"\\n".join(entries)}" | rofi -p "  WiFi" '
                          f'-config {config_file} -dmenu', shell=True,
                          capture_output=True, text=True).stdout.strip()

# Gestion des connexions selon le type (active/connue/nouvelle)
if selected.startswith("") or selected.startswith(""):  # Icônes active/known
    # Menu contextuel : Connect/Disconnect/Show password/Forget
    options = ["Connect", "Show password", "Forget", "Exit"]
    # ... gestion des actions ...
else:
    # Nouvelle connexion : demande de mot de passe
    password = subprocess.run(f'rofi -password -p "Password" -dmenu',
                              shell=True, capture_output=True, text=True)
    subprocess.run(f'nmcli dev wifi con "{selected}" password "{password.stdout}"')

# Notifications de succès/erreur
subprocess.run(f'dunstify -r {dunst_id} -a {dunst_app} -i {success_img} '
               f'"Connected to {selected}"', shell=True)
```

(Script à titre indicatif)

6. Points Techniques Notables et Défis Relevés

6.1. Gestion des Pilotes Graphiques (NVIDIA Hybride)

La configuration des pilotes graphiques NVIDIA sur un ordinateur portable hybride (avec un GPU Intel intégré) a été un défi particulièrement formateur. Cela incluait :

- Installation des pilotes propriétaires NVIDIA ;
- Configuration correcte de Wayland pour gérer le basculement entre les GPU (via des variables d'environnement pour Hyprland, et des configurations spécifiques au gestionnaire d'affichage et applications) ;
- Résolution de problèmes de tearing, de performance, ou de non-reconnaissance du GPU discret.

Cet apprentissage est crucial pour une utilisation stable et performante de Linux sur du matériel moderne.

6.2. Optimisation des Performances des Scripts

La performance des scripts Python pour Rofi est à revoir, j'envisage leur migration vers Bash ou C. J'ai pris en conscience des limitations de performance de Python pour des tâches qui doivent être très réactives (comme l'affichage de menus) et j'ai une volonté d'optimiser cela prochainement.

7. Installation et Dépendances

7.1. Prérequis Système

- **Distribution :** Arch Linux (ou une distribution basée sur Arch compatible avec les paquets nécessaires) ;
- **Accès Sudo :** Nécessaire pour installer des paquets et configurer certains services.

7.2. Paquets Pacman et AUR

Voici les différents paquets à installer pour utiliser les fichiers de configuration mentionnés :

- **Pacman :** firefox, thunar, gvfs, lazygit, fzf, ripgrep, pamixer, unzip, dotnet-runtime, dotnet-sdk, dunst, wl-clipboard, wev, swww, brightnessctl, power-profiles-daemon, ruby, zsh, hyprlock, waybar, neofetch, nwg-look, neovim, curl, git, cliphist, pavucontrol, github-cli, bluez, bluez-utils, blueman, rofi-wayland.
- **AUR (via yay) :** grimblast, webcord, noto-fonts-cjk, ttf-joypixels, spotify-launcher, oh-my-posh, waypaper, catppuccin-gtk-theme-mocha, nbfc-linux.

7.3. Outils Additionnels et Configuration Manuelle

- **Oh My Zsh :** Installation via le script officiel.
    - Plugins zsh-syntax-highlighting et zsh-autosuggestions à cloner manuellement dans le répertoire des plugins Oh My Zsh.

- **ColorLS :** Installation via gem (gestionnaire de paquets Ruby).
- **ASDF Version Manager :** Installation et configuration pour gérer les versions de Node.js.
    - Installation de Node.js via ASDF.
- **FiraCode Nerd Font :** Installation via oh-my-posh font install FiraCode.
- **Copie des Dotfiles :** Les fichiers de configuration du projet (hyprfiles/.config/*) doivent être copiés dans le répertoire ~/.config/ de l'utilisateur.
- **Permissions des Scripts :** Les scripts Bash et Python doivent être rendus exécutables (chmod +x).

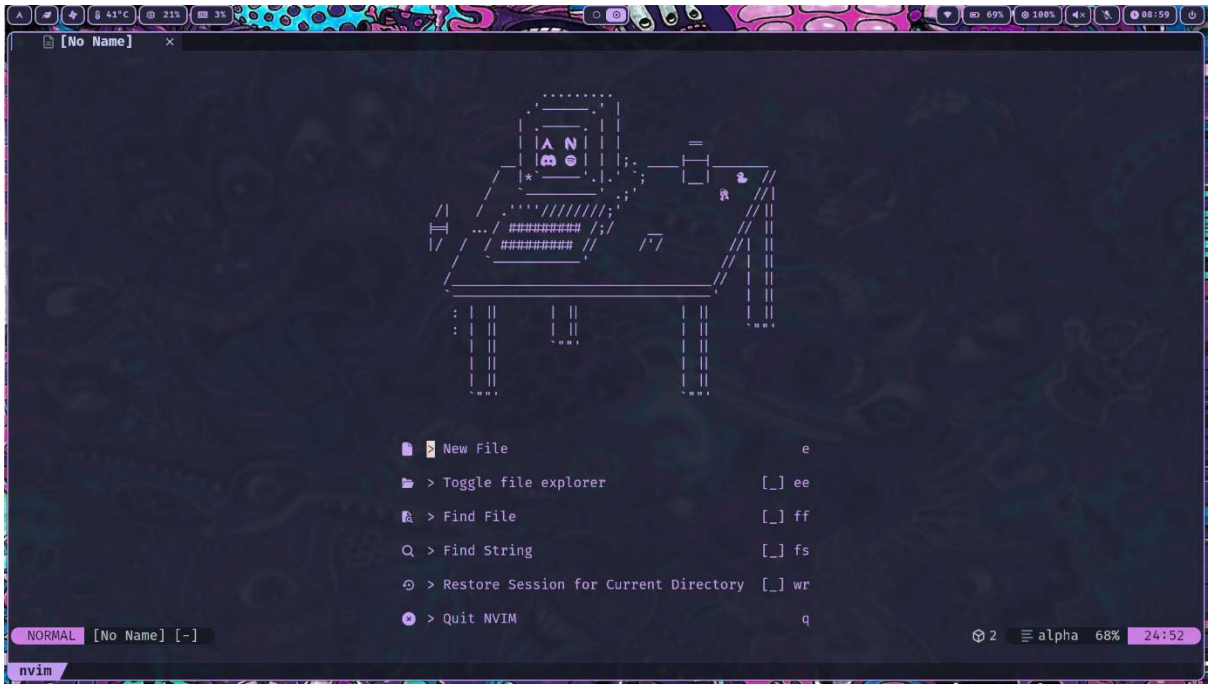
8. Perspectives d'Évolution et Améliorations Futures

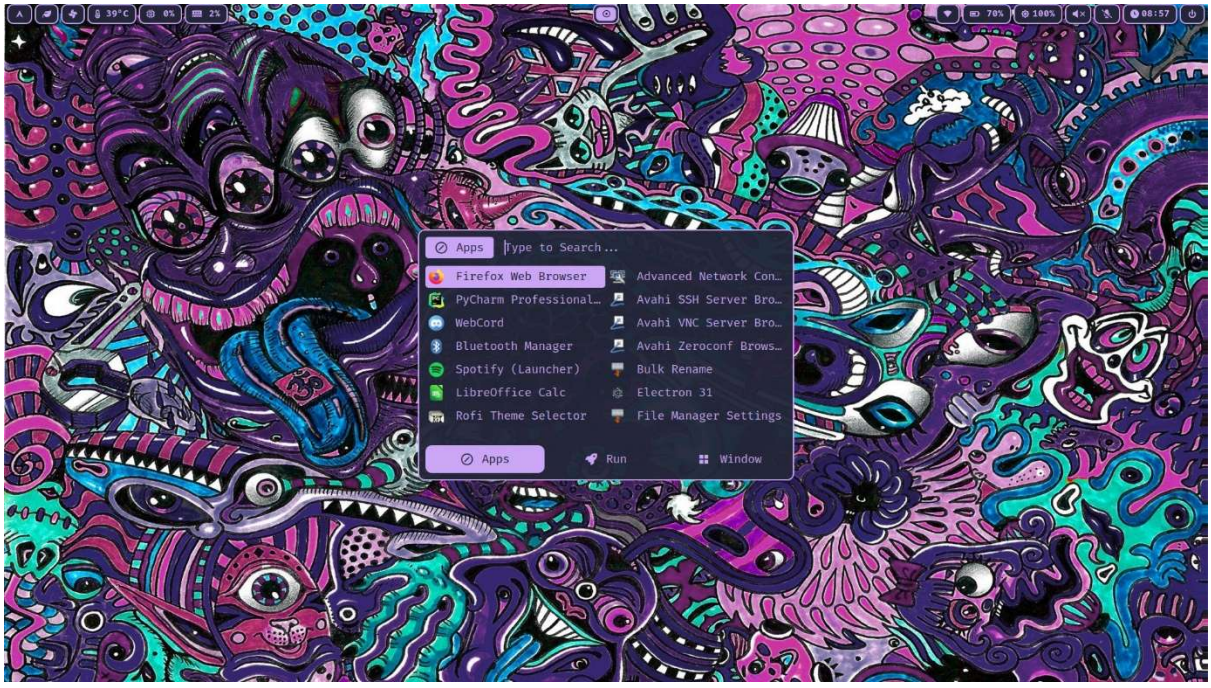
- **Optimisation des Performances des Scripts Rofi :** Remplacer les scripts Python des menus Rofi (fanmenu, wifimenu, powermenu) par des équivalents en Bash, C ou Rust pour une réactivité accrue ;
- **Améliorations du fanmenu :**
    - Ajouter des tooltips pour clarifier la fonction de chaque mode ;
    - Envoyer des notifications dunstify lors d'un changement de mode de ventilation ;
    - Implémenter un mode "personnalisé" permettant à l'utilisateur de définir manuellement une vitesse de ventilateur.
- **Extensions du wifimenu :**
    - Gérer et afficher les informations d'adresse IP, masque de sous-réseau, etc ;
    - Détecter si une connexion Ethernet est active et adapter le menu ou afficher des informations pertinentes.
- **Intégrations Rofi Supplémentaires :**
    - Remplacer l'outil waypaper (gestionnaire de fonds d'écran) par un menu Rofi personnalisé ;
    - Ajouter un sélecteur d'emojis via Rofi pour une insertion facile dans les champs de texte.
- **Centralisation de la Configuration :** Pour les scripts, envisager des fichiers de configuration externes (TOML, YAML) plutôt que des variables en dur ;
- **Amélioration du Logging et Debugging :** Mettre en place des mécanismes de journalisation plus structurés pour les scripts complexes.

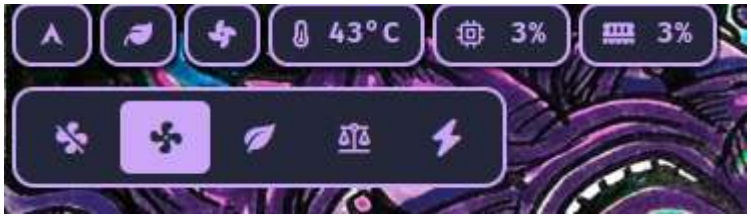
9. Conclusion et Apports Personnels

M'investir dans la personnalisation de mon environnement Arch Linux avec Hyprland m'a beaucoup appris. En configurant chaque détail et en créant mes propres scripts pour des choses comme les menus Rofi ou les notifications, j'ai vraiment appris comment le système

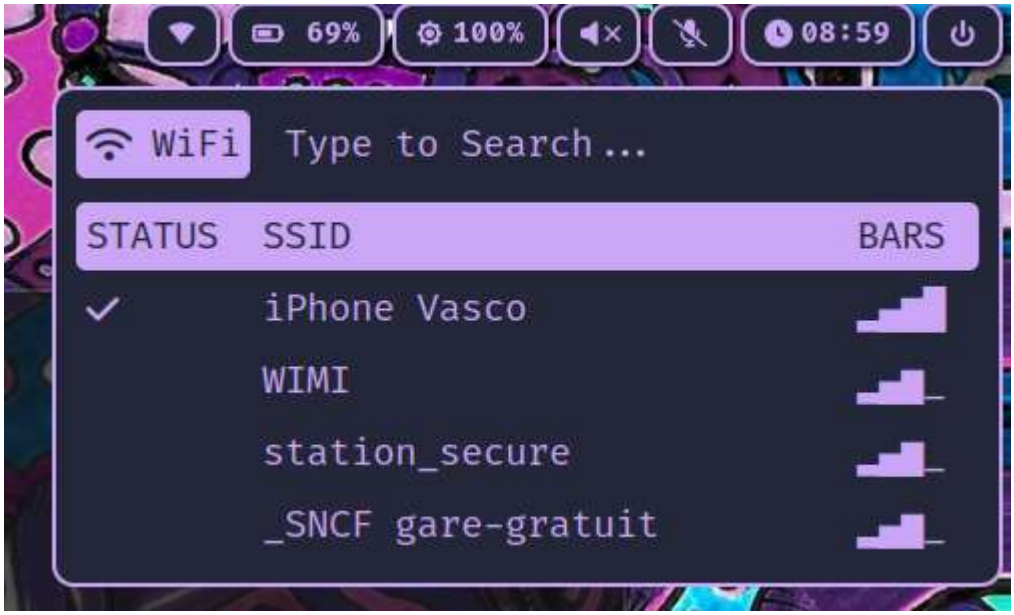
fonctionne de l'intérieur. Résoudre des problèmes complexes, comme celui des pilotes graphiques NVIDIA sur mon pc portable, m’a pris beaucoup de temps et recherches, mais c’était justement ça le but de ce projet ! Le fait d’avoir pu « créer » mon propre système d’exploitation a été une expérience enrichissante que je recommanderai à n’importe quel passionné d’informatique.








67%
100%
09:01


69%
100%
08:59
WiFi
Type to Search ...
STATUS
SSID
BARS
iPhone Vasco
WIMI
station_secure
_SNCF gare-gratuit