



# Projet MiniShell

## 1. Gestion de la date et de l'heure (date.c et date.h)

- **date.c:** Ce fichier implémente une fonction `print_current_date_and_time()` qui utilise la bibliothèque `time.h` pour récupérer la date et l'heure actuelles du système et les afficher. Cette fonction est appelée dans `main()`.
- **date.h:** Ce fichier d'en-tête déclare la fonction `print_current_date_and_time()` pour permettre son utilisation dans d'autres fichiers, assurant ainsi une modularité et une réutilisation du code.

## 2. Listage de répertoire (ls.c et ls.h)

- **ls.c:** Le fichier implémente une fonction `listerRepertoire()` qui liste les fichiers et dossiers d'un répertoire donné. Il supporte les options `a` (ou `-all` ou `A`) pour afficher tous les fichiers (y compris cachés) et `l` pour un affichage détaillé (taille de fichier, etc.). La gestion des erreurs (par exemple, un répertoire inexistant) est aussi prise en compte.
- **ls.h:** Ce fichier d'en-tête déclare la fonction `listerRepertoire()`.

## 3. MiniShell principal (minishell.c et minishell\_functions.h)

- **minishell.c:** C'est le cœur de notre MiniShell. Il utilise d'`eadline` et `history` de la bibliothèque `deadline` pour lire les commandes utilisateur. Le programme gère différentes commandes intégrées (`cd`, `pwd`, `echo`, `exit`, etc.) et enregistre l'historique des commandes. Il supporte également l'exécution de commandes

externes via des fonctions comme `executerLsPersonnalise()`. La gestion des erreurs est présente, notamment via l'utilisation de `errno` et `strerror`.

- **minishell\_functions.h:** Ce fichier contient les déclarations des fonctions et des variables globales utilisées pour structurer le MiniShell, comme `parse_and_execute()` et `executerLsPersonnalise()`.

## 4. Listage des processus (ps.c et ps.h)

- **ps.c:** Ce fichier implémente la commande `listerProcessus()` qui parcourt le répertoire `/proc` pour lister les processus en cours sur le système, identifiés par leur PID. Il affiche chaque PID trouvé.
- **ps.h:** Déclare la fonction `listerProcessus()` pour permettre son utilisation dans le shell.

## 5. Listage des utilisateurs (who.c et who.h)

- **who.c:** Le fichier implémente `listerUtilisateurs()`, qui utilise le fichier système UTM pour lister les utilisateurs actuellement connectés sur la machine. Il lit chaque entrée du fichier `utmp` et affiche le nom d'utilisateur pour les entrées de type `USER_PROCESS`.
- **who.h:** Ce fichier déclare la fonction `listerUtilisateurs()`.

---

## Difficultés rencontrées

- **Gestion des pipes:** La mise en œuvre des pipes pour connecter la sortie d'une commande à l'entrée d'une autre s'est avérée complexe, notamment pour gérer correctement les cas d'arrière-plan. La synchronisation et la gestion des différents états du shell ont représenté un défi important. (Cette partie n'a pas été finalisée).
- **Structure du projet:** La conception initiale de la structure du projet nous a posé quelques problèmes. La structure est devenue assez dense et quelque peu bancale au fur et à mesure que de nouvelles fonctionnalités étaient ajoutées. Cela a nécessité des ajustements constants pour maintenir la cohérence et l'efficacité.