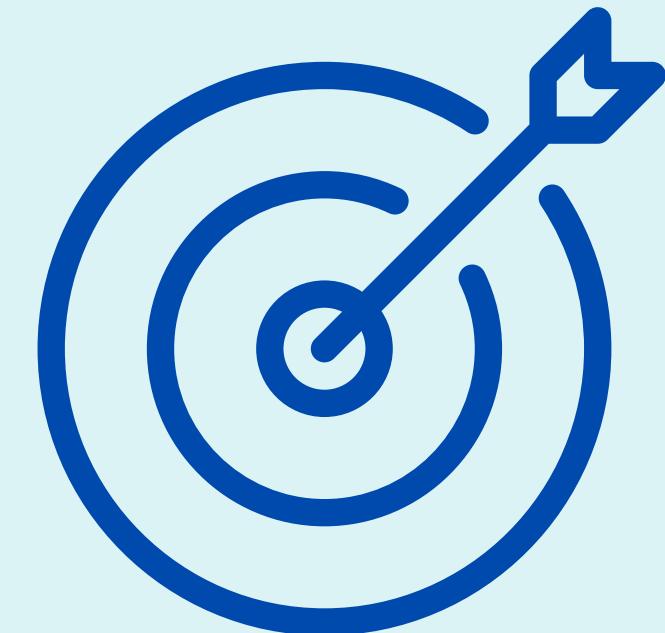




MINISHELL

OBJECTIF ?

**Créer une mini version de bash(shell),
Nous devons apprendre comment bash prend des
arguments, les analyse et les exécute.**

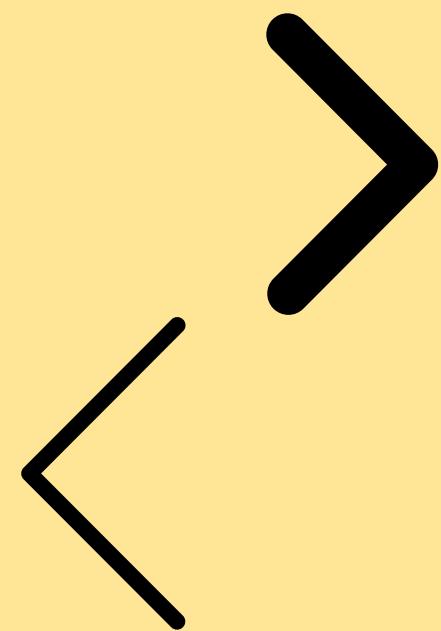
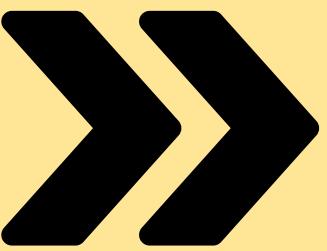


DÉCORTIQUONS Le PROJET



Que doit on faire ?

- **Prompt et Historique** : Le shell doit afficher un prompt pour attendre une nouvelle commande et gérer l'historique des commandes exécutées.
- **Exécution de Commandes** : Il doit chercher et lancer le bon exécutable, soit via le PATH soit via un chemin relatif ou absolu.
- **Gestion des Signaux** : Ton shell doit gérer correctement les signaux comme CTRL-C, CTRL-D, et CTRL-, similairement à Bash.
- **Citations et Caractères Spéciaux** : Il ne doit pas interpréter des guillemets non fermés ou des caractères spéciaux non requis. Il doit traiter différemment les guillemets simples ('') et doubles ("") pour l'interprétation des caractères à l'intérieur.
- **Redirections et Pipes** : Ton shell doit implémenter les redirections (<, >, <<, >>) et les pipes (|) pour lier la sortie d'une commande à l'entrée d'une autre.
- **Variables d'Environnement et Statut de Sortie** : Il doit gérer les variables d'environnement (expansion avec \$) et le statut de sortie de la dernière commande exécutée en avant-plan (\$?).
- **Builtins** : Implémenter des commandes internes (builtins) comme echo, cd, pwd, export, unset, env, et exit.



Que peut-on utiliser ?

- **Fonctions readline :**

- *readline* : Lit une ligne depuis l'entrée standard et la retourne.
- *rl_clear_history* : Efface la liste d'historique de readline.
- *rl_on_new_line* : Prépare readline à lire une entrée sur une nouvelle ligne.
- *rl_replace_line* : Remplace le contenu de la ligne courante du buffer de readline.
- *rl_redisplay* : Met à jour l'affichage pour refléter les changements dans la ligne d'entrée.
- *add_history* : Ajoute l'entrée la plus récente à la liste d'historique de readline.

- **Fonctions d'E/S standard :**

- *printf* : Affiche des données formatées sur la sortie standard (stdout).

- **Fonctions d'E/S de fichier :**

- *write* : Écrit des données dans un descripteur de fichier.
- *access* : Vérifie les permissions du processus appelant pour un fichier ou un répertoire.
- *open* : Ouvre un fichier ou un périphérique, renvoyant un descripteur de fichier.
- *read* : Lit des données depuis un descripteur de fichier dans un tampon.
- *close* : Ferme un descripteur de fichier précédemment ouvert.

- **Fonctions de gestion des erreurs :**

- *strerror* : Renvoie un pointeur vers la représentation textuelle d'un code d'erreur.
- *perror* : Affiche une description textuelle de l'erreur courante sur stderr.

Mais encore ?

- **Fonctions de contrôle de processus :**

- *fork* : Crée un nouveau processus en dupliquant le processus appelant.
- *wait* : Suspend l'exécution du processus appelant jusqu'à ce que l'un de ses enfants se termine.
- *waitpid* : Attends qu'un processus enfant spécifique change d'état.
- *wait3* : Attends qu'un processus enfant quelconque change d'état.
- *wait4* : Attends qu'un processus enfant spécifique change d'état.
- *signal* : Gère ou ignore les signaux envoyés au processus.
- *sigaction* : Gère ou ignore les signaux envoyés au processus.
- *sigemptyset* : Initialise et ajoute des signaux à un ensemble de signaux.
- *sigaddset* : Initialise et ajoute des signaux à un ensemble de signaux.
- *kill* : Envoie un signal à un processus ou à un groupe de processus.
- *exit* : Termine le processus appelant.

- **Fonctions d'allocation de mémoire :**

- *malloc* : Alloue un nombre spécifié d'octets de mémoire sur le tas.
- *free* : Libère la mémoire précédemment allouée.

- **Fonctions de descripteur de fichier :**

- *dup* : Duplique un descripteur de fichier.
- *dup2* : Duplique un descripteur de fichier.
- *pipe* : Crée un pipe pour la communication inter-processus.

Mais encore ?

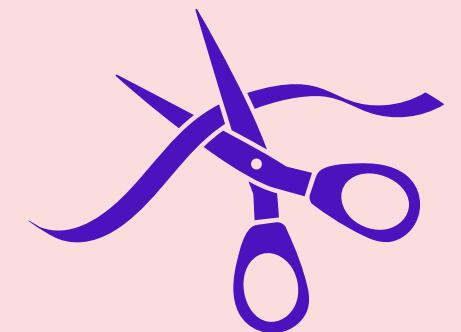
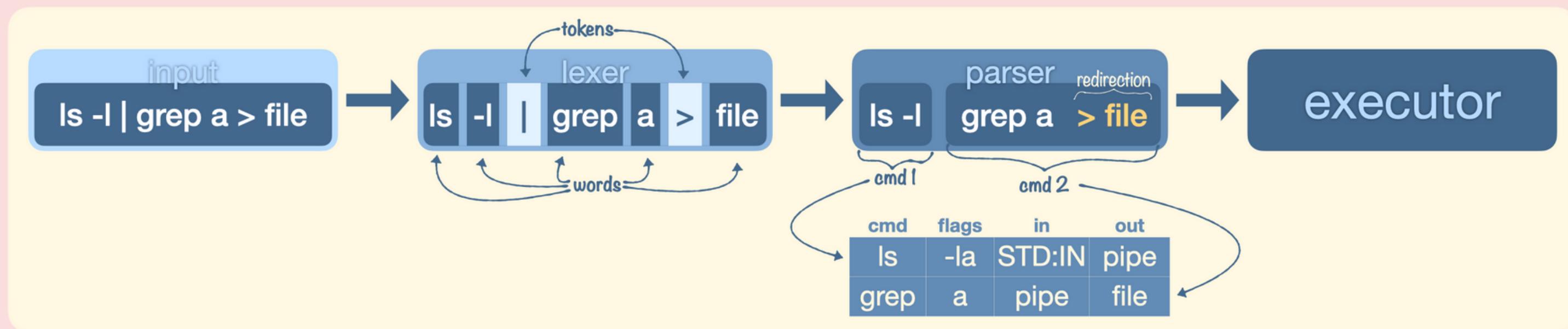
- **Fonctions de répertoire :**

- *getcwd* : Obtient le répertoire de travail actuel.
- *chdir* : Change le répertoire de travail actuel.
- *stat* : Renvoie des informations sur un fichier ou un descripteur de fichier.
- *Istat* : Renvoie des informations sur un fichier ou un descripteur de fichier.
- *fstat* : Renvoie des informations sur un fichier ou un descripteur de fichier.
- *unlink* : Supprime un lien vers un fichier.
- *execve* : Remplace l'image du processus actuel par une nouvelle image de processus.
- *opendir* : Gère les flux de répertoires.
- *readdir* : Gère les flux de répertoires.
- *closedir* : Gère les flux de répertoires.

- **Fonctions de terminal :**

- *isatty* : Teste si un descripteur de fichier fait référence à un terminal.
- *ttynname* : Renvoie le nom du terminal associé à un descripteur de fichier.
- *ttyslot* : Renvoie le nom du terminal associé à un descripteur de fichier.
- *ioctl* : Contrôle les opérations d'entrée/sortie spécifiques à un périphérique.
- *getenv* : Renvoie la valeur d'une variable d'environnement.
- *tcsetattr* : Définit et obtient les attributs d'un terminal.
- *tcgetattr* : Obtient les attributs d'un terminal.
- ...

MAIS COMMENT LE SHELL DÉCOUPE UNE CMD ?



Le PARSING PAR TOKENS

1- Analyse Lexicale (Lexer):

- Son rôle est de lire le texte d'entrée et de le diviser en une séquence de tokens.
- Un token est une chaîne de caractères avec une signification attribuée, comme un mot clé, un identifiant, un symbole spécial (par exemple, +, -, *, /), ou une valeur littérale (comme une chaîne de caractères ou un nombre).
- strtok est votre meilleur ami !

2- Analyse Syntaxique (Parser):

- Son rôle est d'identifier les commandes, les arguments, les redirections, les pipes, etc.
- En général, le premier token est la commande, et les suivants sont ses arguments, jusqu'à ce que vous rencontriez un opérateur de pipe | ou de redirection >, <, >>, <<.
- Si vous rencontrez un opérateur, vous devrez traiter ce qui suit comme une nouvelle commande (dans le cas d'un pipe) ou comme un fichier de redirection.

**EXECVE ET STRTOK TES MEILLEURS
AMIS**



execve

La fonction execve est utilisée pour exécuter un programme. Elle remplace le processus actuel par un nouveau processus spécifié par le chemin d'accès à un fichier exécutable. Les arguments execve sont :

- pathname : le chemin d'accès complet au fichier exécutable que vous souhaitez exécuter.
- argv : un tableau de chaînes de caractères représentant les arguments passés au programme. Le premier argument est généralement le nom du programme.
- envp : un tableau de chaînes de caractères représentant les variables d'environnement pour le nouveau processus.

Quand vous utilisez execve, le processus actuel cesse d'exister et est remplacé par le nouveau programme. Si execve réussit, il ne retourne pas ; si échec, il retourne -1.

STRTOK

La fonction **strtok** est utilisée pour découper une chaîne de caractères en tokens (jetons) selon des délimiteurs spécifiés. Elle est souvent utilisée pour parser des commandes ou des entrées utilisateur. Les arguments **strtok** sont :

- str : la chaîne de caractères à découper lors du premier appel. Pour les appels suivants (pour obtenir d'autres tokens de la même chaîne), passez NULL.
- delim : les délimiteurs utilisés pour découper la chaîne.

strtok modifie la chaîne originale en remplaçant les délimiteurs par \0 (fin de chaîne) et retourne un pointeur vers le premier token trouvé dans la chaîne. Pour obtenir d'autres tokens de la même chaîne, continuez à appeler **strtok** avec **str** NULL.

Contrairement à **split** qui alloue de la mémoire, **strtok** n'en a pas besoin donc moins de fuite de mémoire.

on DÉTAILLE un PEU !

- **prompt** : va voir notre ami `getline()` et `read()`...
- **historique** : il faut stocker tes commandes dans un tableau...
- **path** : vérifie si le fichier est exécutable. Pour les commandes dans PATH, divise PATH en répertoires, recherche l'exécutable dans ces répertoires, et utilise `execve` pour lancer le programme.
- **Les quotes** :
 - simples ' permettent de considérer le contenu comme une chaîne littérale
 - doubles " permettent l'interprétation des variables.
 - A toi de trouver comment il faut les parser. Teste dans le shell.
- **les variable d'environnement** : Utilisez `getenv()`, `setenv()`, `unsetenv()` pour manipuler les variables d'environnement.
- **\$?** : Récupère le statut de sortie de la dernière commande exécutée. Après chaque commande exécutée, stocke le statut de sortie (exit status) pour que \$? puisse le récupérer.
- **contrôle clavier** : Utilise `signal()` ou `sigaction()`. Si tu veux utiliser une globale, c'est maintenant.
- **Les builtins** : `echo`, `cd`, `pwd`, `export`, `unset`, `env`, `exit`.

Les builtins

- **echo avec l'option `-n` :** Affichez les arguments donnés, en omettant le saut de ligne final si `-n` est spécifié.
- **cd :** Utilisez `chdir`. ATTENTION, chemins relatifs et absolus à traiter
- **pwd :** Utilisez `getcwd`.
- **export :** Modifiez les variables d'environnement avec `setenv`.
- **unset :** Supprimez les variables d'environnement avec `unsetenv`.
- **env :** Affichez les variables d'environnement actuelles.
- **exit :** Terminez le shell avec `exit`

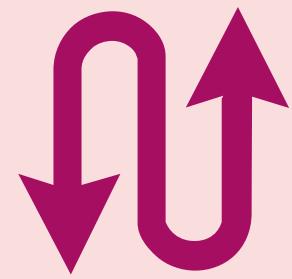
Les builtins 2 Le RETOUR

Nos amis les builtins sont de faux amis, certains semblent simples à implémenter mais ils sont remplis de subtilités.

Notre pote unset en est un parfait exemple !

Donc avant de commencer ,faites un tour d'horizon de votre shell Linux.
Et regardez la correction.

Et **ATTENTION** chaque shell réagit différemment donc TESTEZ sur les ordinateurs de l'école !!!



Les REDIRECTIONS

- < : Redirige l'entrée standard d'un fichier vers un programme.
- > : Redirige la sortie standard d'un programme vers un fichier, en écrasant le fichier.
- << : Heredoc permet l'entrée de plusieurs lignes jusqu'à un délimiteur spécifique.
- >> : Redirige la sortie standard vers un fichier, en ajoutant à la fin du fichier.
- Implémentation : manipulez les descripteurs de fichier (`open`, `dup`, `dup2`) pour rediriger `stdin`, `stdout`, et `stderr`

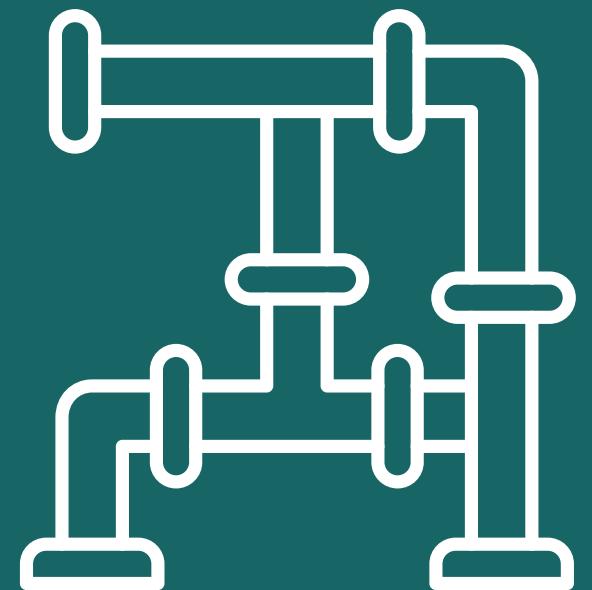


PIPE

Principe fondamental , la sortie d'une commande devient l'entrée d une autre !!!

Il faut maîtriser les processus enfants ;).

Utilisez pipe(), fork(), dup2() pour connecter la sortie d'une commande à l'entrée d'une autre.



TIPS.&.co!

Séparez les cmd avec pipe et les cmd sans pipe !

Et à chaque rajout d'une cmd testez tout tout tout TOUUUUUUUUT ce que vous avez déjà implementé depuis le début (créer son propre testeur est une bonne solution !).

Car souvent, même très souvent, dès que vous rajoutez une cmd, une autre part en



MERCI