Fonctions

Nous pouvons utiliser certaines fonctions que nous connaissons déjà (comme `printf`, `malloc`, `free`, etc.) donc je ne les décrirai pas ici.

Vous n'utiliserez probablement pas toutes ces fonctions, mais au moins vous avez un endroit où vous pouvez facilement trouver des liens vers les pages du manuel. Et pour certains, un exemple sur la façon de les utiliser.

lire la ligne()

```
char *readline (const char *prompt);
```

La `readline()` fonction lit une ligne du terminal et la renvoie en utilisant `prompt` comme invite. Si aucune invite n'est donnée en paramètre, aucune invite ne sera affichée dans le terminal. La ligne renvoyée est allouée avec `malloc` et nous devons la libérer nous-mêmes.

lire la ligne()

```c
#include <stdio.h>
#include <readline/readline.h>
#include <readline/history.h>

int main(void)
{
    char *rl;
    rl = readline("Prompt > ");
    printf("%s\n", rl);
    return (0);
}
```

La compilation et l'exécutio

```
$> ./minishell
Prompt > Hi ! How are you ?
Hi ! How are you ?
$>
```

Vous pouvez trouver plus d'informations `readline()` ici .

rl_clear_history()

```
void rl_clear_history(void);
```

La `rl_clear_line()` fonction efface la liste d'historique en supprimant toutes les entrées. La `rl_clear_line()` fonction libère les données que la `readline` bibliothèque enregistre dans la liste d'historique.

rl_sur_nouvelle_ligne()

```
int rl_on_new_line(void);
```

La `rl_on_new_line()` fonction indique à la routine de mise à jour que nous sommes passés à une nouvelle ligne vide, généralement utilisée après la sortie d'une ligne.

rl_replace_line()

Je n'ai trouvé aucune information sur cette fonction.

rl_redisplay()

```
int rl_redisplay(void);
```

Modifiez `rl_redisplay()` ce qui
`rl_line_buffer` .

ajouter_historique()

```
void add_history(char *s);
```

La `add_history()` fonction enregistre la ligne passée en paramètre dans l'historique afin qu'elle puisse être récupérée ultérieurement dans le terminal (comme appuyer sur la flèche vers le haut dans bash).

obtenircwd()

```
#include <unistd.h>
char *getcwd(char *buf, size_t size);
```

La `getcwd()` fonction renvoie une chaîne terminée par un caractère nul contenant le chemin d'accès absolu qui correspond au répertoire de travail actuel du processus appelant. Le chemin d'accès est renvoyé comme résultat de la fonction et via l'argument `buf` .

> Exemple de getcwd()

Vous pouvez trouver plus d'informations `getcwd()` ici .

chdir()

```
#include <unistd.h>
int chdir(const char *path);
```

`chdir()` modifie le répertoire de travail actuel du processus appelant vers le répertoire spécifié dans `path` .

> exemple chdir()

Vous pouvez trouver plus d'infor

stat() et lstat() et fstat()

```
#include <sys/stat.h>
int stat(const char *restrict pathname, struct stat *restrict statbuf);
int lstat(const char *restrict pathname, struct stat *restrict statbuf);
int fstat(int fd, struct stat *statbuf);
```

Ces fonctions renvoient des informations sur un fichier dans la structure pointée par `statbuf`.

Vous trouverez des informations plus détaillées sur ces fonctions ici .

ouvrir le répertoire()

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir(const char *name);
```

La `opendir()` fonction ouvre un flux de répertoire correspondant au nom du répertoire et renvoie un pointeur vers le flux de répertoire. Le flux est positionné à la première entrée du répertoire.

Vous pouvez trouver plus d'informations sur la `opendir` fonction ici .

liredir()

```
#include <dirent.h>
struct dirent *readdir(DIR *dirp);
```

La `readdir()` fonction renvoie un pointeur vers une `dirent` structure représentant l'entrée de répertoire suivante dans le flux de répertoires pointé par `dirp`. Elle revient `NULL` à la fin du flux de répertoires ou si une e

Vous pouvez trouver plus d'infor

ferméir()

```
#include <sys/types.h>
#include <dirent.h>
int closedir(DIR *dirp);
```

La `closedir()` fonction ferme le flux de répertoire associé à `dirp`. Un appel réussi à `closedir()` ferme également le descripteur de fichier sous-jacent associé à `dirp`. Le descripteur de flux de répertoire `dirp` n'est pas disponible après cet appel.

Vous pouvez trouver plus d'informations `closedir` ici .

erreur()

```
#include <string.h>
char *strerror(int errnum);
```

La `strerror()` fonction renvoie un pointeur vers une chaîne qui décrit le code d'erreur passé dans l'argument errnum. Cette chaîne ne doit pas être modifiée par l'application, mais peut être modifiée par un appel ultérieur à `strerror()` ou `strerror_l()`. Aucune autre fonction de bibliothèque, y compris `perror()`, ne modifiera cette chaîne.

Vous pouvez trouver plus d'informations `strerror` ici .

erreur()

```
#include <stdio.h>
void perror(const char *s);
```

La `perror()` fonction produit un message sur l'erreur standard décrivant la dernière erreur rencontrée lors d'un appel à une fonction système ou de bibliothèque.

Vous pouvez trouver plus d'informations `perror` ici .

estatty()

```
#include <unistd.h>
int isatty(int fd);
```

La `isatty` fonction teste s'il s'agit `fd` d'un terminal.

Vous pouvez trouver plus d'informations `isatty` ici .

nom_tty()

```
#include <unistd.h>
char **ttyname(int fd);
```

La `ttyname()` fonction renvoie un pointeur vers le chemin d'accès terminé par null du périphérique terminal ouvert sur le descripteur de fichier `fd` , ou `NULL` en cas d'erreur.

Vous pouvez trouver plus d'informations `ttyname()` ici .

ttyslot()

```
#include <unistd.h>
int ttyslot(void);
```

Il s'agit d'une fonction héritée avec une certaine histoire, vous pouvez tout lire à ce sujet et comment elle fonctionne ici .

ioctl()

```
#include <sys/ioctl.h>
int ioctl(int fd, unsigned long request, ...);
```

L' `ioctl()` appel système manipule les paramètres de périphérique sous-jacents d'un fichier spécial. Vous trouverez des informations plus détaillées ici .

getenv()

```
#include <stdlib.h>
char *getenv(const char *na
```

La `getenv()` fonction recherche dans la liste d'environnements pour trouver le nom de la variable d'environnement et renvoie un pointeur vers la chaîne de valeur correspondante.

Vous pouvez trouver plus d'informations `getenv()` ici .

tcsetattr()

```
#include <termios.h>
int tcsetattr(int fildes, int optional_actions, const struct *termios_p);
```

La `tcsetattr()` fonction doit définir les paramètres associés au terminal référencé par le descripteur de fichier ouvert `fildes` à partir de la `termios` structure référencée par `termios_p` comme décrit ici .

tcgetattr()

```
#include <termios.h>
int tcgetattr(int fildes, struct termios *termios_p);
```

La `tcgetattr()` fonction doit obtenir les paramètres associés au terminal référencé par `fildes` et les stocker dans la `termios` structure référencée par `termios_p` .

Vous trouverez des informations plus détaillées ici .

tgetent()

```
#include <curses.h>
#include <term.h>
int tgetent(char *bp, const char *name);
int tgetflag(char *id);
int tgetnum(char *id);
char *tgetstr(char *id, cha
char *tgoto(const char *cap
int tputs(const char *str,
```


Ces routines sont incluses comm
`termcap` bibliothèque. Vous pou

Précédent
Comprendre Minishell

Suivant
Construire la chose

Dernière mise à jourIl y a 5 mois