42sh

Generated by Doxygen 1.13.2

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 aliases_s Struct Reference

`#include <meowsh.h>`

Public Attributes

- char * original
- char * new_name
- bool browsed
- struct aliases_s * next

3.1.1 Member Data Documentation

3.1.1.1 browsed

```
bool aliases_s::browsed
```

3.1.1.2 new_name

```
char* aliases_s::new_name
```

3.1.1.3 next

```
struct aliases_s* aliases_s::next
```

3.1.1.4 original

```
char* aliases_s::original
```

The documentation for this struct was generated from the following file:

- include/meowsh.h

3.2 commands_s Struct Reference

`#include <meowsh.h>`

Public Attributes

- sep_t separator
- int depth
- int fd_input
- int fd_output
- char * heredoc
- char * path
- char ** command
- struct commands_s ** subcommands

3.2.1 Member Data Documentation

3.2.1.1 command

```
char** commands_s::command
```

3.2.1.2 depth

```
int commands_s::depth
```

3.2.1.3 fd_input

```
int commands_s::fd_input
```

3.2.1.4 fd_output

```
int commands_s::fd_output
```

3.2.1.5 heredoc

```
char* commands_s::heredoc
```

3.2.1.6 path

```
char* commands_s::path
```

3.2.1.7 separator

```
sep_t commands_s::separator
```

3.2.1.8 subcommands

```
struct commands_s** commands_s::subcommands
```

The documentation for this struct was generated from the following file:

- include/meowsh.h

3.3 env_s Struct Reference

```
#include <meowsh.h>
```

Public Attributes

- char ** local
- char ** global

3.3.1 Member Data Documentation

3.3.1.1 global

```
char** env_s::global
```

3.3.1.2 local

```
char** env_s::local
```

The documentation for this struct was generated from the following file:

- include/meowsh.h

3.4 errno_outputs Struct Reference

```
#include <error_outputs.h>
```

Public Attributes

- int errnum
- const char * errmsg

3.4.1 Member Data Documentation

3.4.1.1 errmsg

```
const char* errno_outputs::errmsg
```

3.4.1.2 errnum

```
int errno_outputs::errnum
```

The documentation for this struct was generated from the following file:

- include/error_outputs.h

3.5 exec_data_s Struct Reference

```
#include <exec.h>
```

Public Attributes

- int nb_elt
- int ** pipe
- int * pid
- int * status
- int * cmd_type
- int prev_depth

3.5.1 Member Data Documentation

3.5.1.1 cmd_type

```
int* exec_data_s::cmd_type
```

3.5.1.2 nb_elt

```
int exec_data_s::nb_elt
```

3.5.1.3 pid

```
int* exec_data_s::pid
```

3.5.1.4 pipe

```
int** exec_data_s::pipe
```

3.5.1.5 prev_depth

```
int exec_data_s::prev_depth
```

3.5.1.6 status

```
int* exec_data_s::status
```

The documentation for this struct was generated from the following file:

- include/exec.h

3.6 input_s Struct Reference

```
#include <input.h>
```

Public Attributes

- char * str_input
- char char_input
- size_t input_len
- size_t cursor_pos
- int history_pos
- char * stock_history

3.6.1 Member Data Documentation

3.6.1.1 char_input

```
char input_s::char_input
```

3.6.1.2 cursor_pos

```
size_t input_s::cursor_pos
```

3.6.1.3 history_pos

```
int input_s::history_pos
```

3.6.1.4 input_len

```
size_t input_s::input_len
```

3.6.1.5 stock_history

```
char* input_s::stock_history
```

3.6.1.6 str_input

```
char* input_s::str_input
```

The documentation for this struct was generated from the following file:

- include/input.h

3.7 parsing_state_s Struct Reference

```
#include <parser.h>
```

Public Attributes

- char * line
- sep_t separator
- int current_depth
- bool in_single_quote
- bool in_double_quote
- bool in_backtick
- int start_index
- int current_subcommand_index

3.7.1 Member Data Documentation

3.7.1.1 current_depth

```
int parsing_state_s::current_depth
```

3.7.1.2 current_subcommand_index

```
int parsing_state_s::current_subcommand_index
```

3.7.1.3 in_backtick

```
bool parsing_state_s::in_backtick
```

3.7.1.4 in_double_quote

```
bool parsing_state_s::in_double_quote
```

3.7.1.5 in_single_quote

```
bool parsing_state_s::in_single_quote
```

3.7.1.6 line

```
char* parsing_state_s::line
```

3.7.1.7 separator

```
sep_t parsing_state_s::separator
```

3.7.1.8 start_index

```
int parsing_state_s::start_index
```

The documentation for this struct was generated from the following file:

- include/parser.h

3.8 shell_s Struct Reference

```
#include <meowsh.h>
```

Public Attributes

- commands_t ** commands
- env_t * env
- aliases_t ** aliases
- char ** history
- int history_size
- char * owd
- int exit_status
- bool run

3.8.1 Member Data Documentation

3.8.1.1 aliases

```
aliases_t** shell_s::aliases
```

3.8.1.2 commands

```
commands_t** shell_s::commands
```

3.8.1.3 env

```
env_t* shell_s::env
```

3.8.1.4 exit_status

```
int shell_s::exit_status
```

3.8.1.5 history

```
char** shell_s::history
```

3.8.1.6 history_size

```
int shell_s::history_size
```

3.8.1.7 owd

```
char* shell_s::owd
```

3.8.1.8 run

```
bool shell_s::run
```

The documentation for this struct was generated from the following file:

- include/meowsh.h

3.9 tab_builtins_t Struct Reference

```
#include <meowsh.h>
```

Public Attributes

- char * command
- int(* builtin_func)(shell_t *, char **command, int local)
- int __attribute_maybe_unused__ local

3.9.1 Member Data Documentation

3.9.1.1 builtin_func

```
int(* tab_builtins_t::builtin_func) (shell_t *, char **command, int local)
```

3.9.1.2 command

```
char* tab_builtins_t::command
```

3.9.1.3 local

```
int __attribute_maybe_unused__ tab_builtins_t::local
```

The documentation for this struct was generated from the following file:

- include/meowsh.h

Chapter 4

File Documentation

4.1 include/builtins.h File Reference

```
#include "meowsh.h"
#include "utilities.h"
#include <errno.h>
#include <sys/stat.h>
#include <fcntl.h>
```

Macros

- #define NB_BUILT_IN 12
- #define BUFFER_SIZE 65536

Functions

- int interpret_alias (shell_t *shell, char **command, __attribute_maybe_unused__ int local)

  Handles the `alias` command in the shell.
- int interpret_unalias (shell_t *shell, char **command, __attribute_maybe_unused__ int local)

  Interpret the `unalias` command.
- void add_node (aliases_t **aliases, aliases_t *new_node)

  Add a new alias to the aliases linked list.
- void delete_node (aliases_t **aliases, aliases_t *node, aliases_t *previous)

  Delete an alias from the aliases linked list.
- void free_node (aliases_t *node)

  Free memory allocated for an alias node.
- void get_history (shell_t *shell)

  Load the history into the shell.
- void add_to_history (char *command, shell_t *shell)

  Add a command to the history and save it to the file.
- void free_history (char **history, int count)

  Free a partially filled history.
- int display_history (shell_t *shell, __attribute_maybe_unused__ char **command, __attribute_maybe_↩ unused__ int local)

Display the entire history.

- char * exec_history (shell_t *shell, char *command)
  Execute a command from the history.
- int my_unsetenv (shell_t *sh, char **command, int local)
  Unsets environment variables.
- int my_env (shell_t *sh, char **command, int local)
  Display the environment variables.
- int my_setenv (shell_t *sh, char **command, int local)
  Handles the `setenv` builtin command.
- int add_elt (char ***array, char *elt)
  Adds a new element to the environment array.
- char * handle_tilde (char *directory, char **env, char *previous_dir)
  Handles tilde (~) and other special cases for the `cd` command.
- char * get_root_home (char *user)
  Retrieves the home directory of a specified user.
- int my_cd (shell_t *shell, char **command_args, __attribute_maybe_unused__ int local)
  Handles the `cd` command.
- char * get_value_key (char *key, char **env)
  Retrieves the value associated with a given key in the environment.
- int interpret_which (shell_t *shell, char **command, __attribute_maybe_unused__ int local)
  Handle the `which` command.
- int interpret_where (shell_t *shell, char **command, __attribute_maybe_unused__ int local)
  Handle the `where` command.
- int is_a_builtin_command_where (char *command)
  Check if a command is a shell built-in for where.
- int is_a_builtin_command_which (char *command)
  Check if a command is a shell built-in for which.
- int my_exit (shell_t *shell, char **command, __attribute_maybe_unused__ int local)
  Handles the `exit` command.

4.1.1 Macro Definition Documentation

4.1.1.1 BUFFER_SIZE

```
#define BUFFER_SIZE 65536
```

4.1.1.2 NB_BUILT_IN

```
#define NB_BUILT_IN 12
```

4.1.2 Function Documentation

4.1.2.1 add_elt()

```
int add_elt (
             char *** array,
             char * elt)
```

Adds a new element to the environment array.

This function appends a new environment variable to the given environment array.

Parameters

array	Pointer to the environment array.
elt	The environment variable to add.

Returns

int Returns 0 on success, or 1 on failure.

4.1.2.2 add_node()

```
void add_node (
            aliases_t ** aliases,
            aliases_t * new_node)
```

Add a new alias to the aliases linked list.

This function adds a new alias node to the linked list of aliases. The list is maintained in lexicographical order based on the alias name.

Parameters

aliases	Pointer to the head of the aliases linked list.
new_node	Pointer to the new alias node to add.

4.1.2.3 add_to_history()

```
void add_to_history (
            char * command,
            shell_t * shell)
```

Add a command to the history and save it to the file.

This function adds a command to the shell's history array and appends it to the `.42sh_history` file.

Parameters

command	The command to add to the history.
shell	The shell structure containing the history.

4.1.2.4 delete_node()

```
void delete_node (
            aliases_t ** aliases,
            aliases_t * current,
            aliases_t * previous)
```

Delete an alias from the aliases linked list.

This function removes an alias node from the linked list of aliases and frees its memory. If the alias to delete is the head of the list, the head pointer is updated.

Parameters

aliases	Pointer to the head of the aliases linked list.
current	Pointer to the alias node to delete.
previous	Pointer to the alias node preceding the one to delete.

4.1.2.5 display_history()

```
int display_history (
            shell_t * shell,
            __attribute_maybe_unused__ char ** command,
            __attribute_maybe_unused__ int local)
```

Display the entire history.

This function prints the entire history with an index for each command.

Parameters

shell	The shell structure containing the history.
command	Unused parameter.
local	Unused parameter.

Returns

Always returns 0.

4.1.2.6 exec_history()

```
char * exec_history (
            shell_t * shell,
            char * command)
```

Execute a command from the history.

This function searches the history for a command matching the given input and returns it for execution.

Parameters

shell	The shell structure containing the history.
command	The input string specifying the command to execute.

Returns

A string containing the command from the history, or NULL if not found.

4.1.2.7 free_history()

```
void free_history (
            char ** history,
            int count)
```

Free a partially filled history.

This function frees the memory allocated for a partially filled history array, up to the specified count.

Parameters

history	The history array.
count	Number of valid entries to free.

4.1.2.8 free_node()

```
void free_node (
            aliases_t * node)
```

Free memory allocated for an alias node.

This function frees the memory allocated for a single alias node, including its `original` and `new_name` strings.

Parameters

node	Pointer to the alias node to free.

4.1.2.9 get_history()

```
void get_history (
            shell_t * shell)
```

Load the history into the shell.

This function loads the history from the `.42sh_history` file into the shell's history array.

Parameters

shell	The shell structure containing the history.

4.1.2.10 get_root_home()

```
char * get_root_home (
            char * user)
```

Retrieves the home directory of a specified user.

This function reads the `/etc/passwd` file, searches for the specified user, and extracts their home directory from the corresponding line.

Parameters

user	The username whose home directory is to be retrieved.

Returns

A dynamically allocated string containing the user's home directory, or NULL if the user is not found or an error occurs.

4.1.2.11 get_value_key()

```
char * get_value_key (
            char * key,
            char ** env)
```

Retrieves the value associated with a given key in the environment.

This function iterates through the environment variables to find a key that matches the specified key and returns its value.

Parameters

key	The key to search for in the environment variables.
env	The array of environment variables.

Returns

A pointer to the value associated with the key, or NULL if the key is not found.

4.1.2.12 handle_tilde()

```
char * handle_tilde (
            char * directory,
            char ** env,
            char * previous_dir)
```

Handles tilde (~) and other special cases for the `cd` command.

This function processes the input directory string and handles:

- ~ for the current user's home directory.
- ~`username` for another user's home directory.

Parameters

directory	The input directory string.
env	The environment variables array.
previous_dir	The previous directory string.

Returns

A newly allocated string containing the processed directory.

4.1.2.13 interpret_alias()

```
int interpret_alias (
            shell_t * shell,
            char ** command,
            __attribute_maybe_unused__ int local)
```

Handles the `alias` command in the shell.

This function interprets the `alias` command based on the number of arguments provided:

- If no arguments are provided, it displays all existing aliases.
- If one argument is provided, it does nothing (reserved for future use).
- If two or more arguments are provided, it adds a new alias to the list.

Parameters

shell	The shell structure containing the alias list.
command	The array of arguments passed to the `alias` command.
local	Unused parameter (reserved for future use).

Returns

0 on success, 1 on failure.

4.1.2.14 interpret_unalias()

```
int interpret_unalias (
            shell_t * shell,
            char ** command,
            __attribute_maybe_unused__ int local)
```

Interpret the `unalias` command.

This function processes the `unalias` command entered by the user. It checks if the command has enough arguments and deletes the specified aliases from the shell's alias list. If no arguments are provided, an error message is displayed.

Parameters

shell	Pointer to the shell structure containing the alias list.
command	Array of strings representing the `unalias` command and its arguments.
local	Unused parameter for compatibility with other builtins.

Returns

Exit code: 0 on success, 1 on failure (e.g., too few arguments).

4.1.2.15 interpret_where()

```
int interpret_where (
            shell_t * shell,
            char ** command,
            __attribute_maybe_unused__ int local)
```

Handle the `where` command.

This function processes the `where` command, which identifies all locations of executables or aliases. It checks if the provided arguments are aliases or commands in the `PATH` and prints all their locations.

Parameters

shell	Pointer to the shell structure containing aliases and environment variables.
command	Array of strings representing the `where` command and its arguments.
local	Unused parameter for compatibility with other builtins.

Returns

0 on success, 1 on failure (e.g., too few arguments).

4.1.2.16 interpret_which()

```
int interpret_which (
            shell_t * shell,
            char ** command,
            __attribute_maybe_unused__ int local)
```

Handle the `which` command.

This function processes the `which` command, which identifies the location of executables or aliases. It checks if the provided arguments are aliases or commands in the `PATH` and prints their locations.

Parameters

shell	Pointer to the shell structure containing aliases and environment variables.
command	Array of strings representing the `which` command and its arguments.
local	Unused parameter for compatibility with other builtins.

Returns

0 on success, 1 on failure (e.g., too few arguments).

4.1.2.17 is_a_builtin_command_where()

```
int is_a_builtin_command_where (
            char * command)
```

Check if a command is a shell built-in for where.

This function loops through the `tab` of built-in commands and checks if the given command matches any of the built-ins. If it is a built-in, it prints a message indicating that it is a shell built-in command.

Parameters

command	The command to check.

Returns

1 if the command is a built-in, 0 otherwise.

4.1.2.18 is_a_builtin_command_which()

```
int is_a_builtin_command_which (
            char * command)
```

Check if a command is a shell built-in for which.

This function loops through the `tab` of built-in commands and checks if the given command matches any of the built-ins. If it is a built-in, it prints a message indicating that it is a shell built-in command.

Parameters

command	The command to check.

Returns

1 if the command is a built-in, 0 otherwise.

4.1.2.19 my_cd()

```
int my_cd (
              shell_t * shell,
              char ** command_args,
              __attribute_maybe_unused__ int local)
```

Handles the `cd` command.

This function processes the `cd` command, handling arguments, tilde expansion, and error cases such as too many arguments or invalid paths. It updates the shell's old working directory (`owd`) and switches to the specified directory.

Parameters

shell	The shell structure containing environment variables and the old working directory.
command_args	The arguments passed to the `cd` command.
local	Unused parameter for compatibility with other builtins.

Returns

0 on success, 1 on failure.

4.1.2.20 my_env()

```
int my_env (
              shell_t * sh,
              char ** command,
              int local)
```

Display the environment variables.

This function prints the environment variables stored in the shell's global or local environment, depending on the `local` parameter. If additional arguments are passed to the `env` command, an error message is displayed, and the shell's exit status is set to `127`.

Parameters

sh	Pointer to the shell structure containing environment variables.
command	Array of strings representing the `env` command and its arguments.
local	If set to `1`, displays the local environment; otherwise, displays the global environment.

Returns

Always returns `0`.

4.1.2.21 my_exit()

```
int my_exit (
            shell_t * shell,
            char ** command,
            __attribute_maybe_unused__ int local)
```

Handles the `exit` command.

This function processes the `exit` command, which terminates the shell. If an exit code is provided as an argument, it is used as the shell's exit code. Otherwise, the shell's current `exit_status` is used.

Parameters

shell	Pointer to the shell structure.
command	Array of strings representing the `exit` command and its arguments.
local	Unused parameter for compatibility with other builtins.

Returns

The exit code to terminate the shell with.

4.1.2.22 my_setenv()

```
int my_setenv (
            shell_t * shell,
            char ** command,
            int local)
```

Handles the `setenv` builtin command.

This function implements the logic for the `setenv` command, which sets or modifies environment variables. It validates the input, checks for errors, and updates the appropriate environment (local or global).

Parameters

shell	The shell structure containing the environment.
command	The command arguments, where `command[1]` is the variable name and `command[2]` is the value (optional).
local	A flag indicating whether to modify the local or global environment.

Returns

The correct exit status.

4.1.2.23 my_unsetenv()

```
int my_unsetenv (
            shell_t * sh,
            char ** command,
            int local)
```

Unsets environment variables.

This function implements the `unsetenv` builtin. It removes each environment variable provided in the command arguments. If no variable is given, it writes an error message and sets the shell's exit status.

Parameters

sh	Pointer to the shell structure.
command	Array of command arguments.
local	Non-zero for the local environment, zero for global.

Returns

int Returns 1 if an error occurred, otherwise 0.

4.2 builtins.h

Go to the documentation of this file.

```
00001 /*
00002 ** EPITECH PROJECT, 2025
00003 ** 42sh
00004 ** File description:
00005 ** builtins header
00006 */
00007
00008 #ifndef BUILTINS
00009     #define BUILTINS
00010     #include "meowsh.h"
00011     #include "utilities.h"
00012     #include <errno.h>
00013     #include <sys/stat.h>
00014     #include <fcntl.h>
00015     #define NB_BUILT_IN 12
00016     #define BUFFER_SIZE 65536
00017
00018
00019 // Alias
00020 int interpret_alias(shell_t *shell, char **command,
00021     __attribute_maybe_unused__ int local);
00022 int interpret_unalias(shell_t *shell, char **command,
00023     __attribute_maybe_unused__ int local);
00024 void add_node(aliases_t **aliases, aliases_t *new_node);
00025 void delete_node(aliases_t **aliases, aliases_t *node, aliases_t *previous);
00026 void free_node(aliases_t *node);
00027
00028 // History
00029 void get_history(shell_t *shell);
00030 void add_to_history(char *command, shell_t *shell);
00031 void free_history(char **history, int count);
00032 int display_history(shell_t *shell, __attribute_maybe_unused__ char **command,
00033     __attribute_maybe_unused__ int local);
00034 char *exec_history(shell_t *shell, char *command);
00035
00036 // Environment
00037 int my_unsetenv(shell_t *sh, char **command, int local);
00038 int my_env(shell_t *sh, char **command, int local);
00039 int my_setenv(shell_t *sh, char **command, int local);
00040 int add_elt(char ***array, char *elt);
00041
00042 // Cd
00043 char *handle_tilde(char *directory, char **env, char *previous_dir);
00044 char *get_root_home(char *user);
00045 int my_cd(shell_t *shell, char **command_args,
00046     __attribute_maybe_unused__ int local);
00047 char *get_value_key(char *key, char **env);
00048
00049 // Which and where
00050 int interpret_which(shell_t *shell, char **command,
00051     __attribute_maybe_unused__ int local);
00052 int interpret_where(shell_t *shell, char **command,
00053     __attribute_maybe_unused__ int local);
00054 int is_a_builtin_command_where(char *command);
00055 int is_a_builtin_command_which(char *command);
00056
00057 // Exit
00058 int my_exit(shell_t *shell, char **command,
00059     __attribute_maybe_unused__ int local);
00060
00061 #endif
```

4.3 include/error_outputs.h File Reference

```
#include <errno.h>
#include <stdlib.h>
```

Classes

- struct errno_outputs

Typedefs

- typedef struct errno_outputs errno_outputs_t

Variables

- errno_outputs_t const errno_table [ ]

4.3.1 Typedef Documentation

4.3.1.1 errno_outputs_t

```
typedef struct errno_outputs errno_outputs_t
```

4.3.2 Variable Documentation

4.3.2.1 errno_table

```
errno_outputs_t const errno_table[]
```

Initial value:

```
= {
    {E2BIG, "Argument list too long."},
    {EACCES, "Permission denied."},
    {EFAULT, "Bad address."},
    {EINVAL, "Invalid argument."},
    {EIO, "Input/output error."},
    {EISDIR, "Is a directory."},
    {ELIBBAD, "ELF interpreter format error."},
    {ELOOP, "Too many levels of symbolic links."},
    {EMFILE, "Too many open files."},
    {ENAMETOOLONG, "File name too long."},
    {ENFILE, "Too many open files in system."},
    {ENOENT, "No such file or directory."},
    {ENOEXEC, "Exec format error. Binary file not executable."},
    {ENOMEM, "Not enough memory."},
    {ENOTDIR, "Not a directory."},
    {EPERM, "Operation not permitted."},
    {ETXTBSY, "Text file busy."},
    {-1, NULL}
}
```

4.4 error_outputs.h

Go to the documentation of this file.

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** error ouptus tab
00004 ** File description:
00005 ** my_constants definition
00006 */
00007
00008 #ifndef MY_CONSTANTS
00009     #define MY_CONSTANTS
00010     #include <errno.h>
00011     #include <stdlib.h>
00012
00013 typedef struct errno_outputs {
00014     int errnum;
00015     const char *errmsg;
00016 } errno_outputs_t;
00017
00018 errno_outputs_t const errno_table[] = {
00019     {E2BIG, "Argument list too long."},
00020     {EACCES, "Permission denied."},
00021     {EFAULT, "Bad address."},
00022     {EINVAL, "Invalid argument."},
00023     {EIO, "Input/output error."},
00024     {EISDIR, "Is a directory."},
00025     {ELIBBAD, "ELF interpreter format error."},
00026     {ELOOP, "Too many levels of symbolic links."},
00027     {EMFILE, "Too many open files."},
00028     {ENAMETOOLONG, "File name too long."},
00029     {ENFILE, "Too many open files in system."},
00030     {ENOENT, "No such file or directory."},
00031     {ENOEXEC, "Exec format error. Binary file not executable."},
00032     {ENOMEM, "Not enough memory."},
00033     {ENOTDIR, "Not a directory."},
00034     {EPERM, "Operation not permitted."},
00035     {ETXTBSY, "Text file busy."},
00036     {-1, NULL}
00037 };
00038
00039 #endif /* !MY_CONSTANTS */
```

4.5 include/exec.h File Reference

```
#include "meowsh.h"
#include "builtins.h"
#include <sys/types.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <string.h>
#include <dirent.h>
```

Classes

- struct exec_data_s

Macros

- #define CMD 0
- #define BUILT_CMD 1
- #define SUBCMD 2
- #define NOTHING 3

Typedefs

- typedef struct exec_data_s exec_data_t

Functions

- void exec_cmd (shell_t *shell, commands_t **cmds, int prev_depth)

  Executes a sequence of commands, handling pipes and separators.
- void check_signal (shell_t *shell, exec_data_t *exec_data, int i)

  Updates the shell's exit status based on the process termination status.
- char * get_cmd_path (char *command, char **env)

  Get the full path of a command.
- int get_cmd_type (commands_t *cmd)

  Determines the type of a given command.
- char ** get_paths (char **env)

  Retrieve the list of directories from the `PATH` environment variable.
- int check_path (char *command, char *path)

  Check if a command exists in a given directory.
- int init_exec_data (commands_t **cmds, exec_data_t *exec_data)

  Initializes the execution data structure.
- void free_exec_data (exec_data_t *exec_data)

  Frees the memory allocated for the execution data structure.
- void manage_saved_fd (int saved[2])

  Restores the saved file descriptors for standard input and output.

4.5.1 Macro Definition Documentation

4.5.1.1 BUILT_CMD

```
#define BUILT_CMD 1
```

4.5.1.2 CMD

```
#define CMD 0
```

4.5.1.3 NOTHING

```
#define NOTHING 3
```

4.5.1.4 SUBCMD

```
#define SUBCMD 2
```

4.5.2 Typedef Documentation

4.5.2.1 exec_data_t

```
typedef struct exec_data_s exec_data_t
```

4.5.3 Function Documentation

4.5.3.1 check_path()

```
int check_path (
            char * command,
            char * path)
```

Check if a command exists in a given directory.

This function searches for a command in the specified directory by iterating through the directory's contents.

Parameters

command	The name of the command to search for.
path	The directory path to search in.

Returns

1 if the command is found, 0 otherwise.

4.5.3.2 check_signal()

```
void check_signal (
            shell_t * shell,
            exec_data_t * exec_data,
            int i)
```

Updates the shell's exit status based on the process termination status.

If the process was a built-in command, it directly sets the shell's exit status to the process's exit code. Otherwise, it updates the exit status based on whether the process exited normally or was terminated by a signal.

Parameters

shell	The shell structure containing the exit status.
status	The status of the terminated process.
is_built_cmd	A boolean indicating if the process was a built-in command.

4.5.3.3 exec_cmd()

```
void exec_cmd (
            shell_t * shell,
            commands_t ** cmds,
            int prev_depth)
```

Executes a sequence of commands, handling pipes and separators.

Initializes execution data, processes each command, and cleans up resources.

Parameters

shell	Pointer to the shell structure.
cmds	Array of commands to execute.
prev_depth	Depth of the previous command in the command tree.

4.5.3.4 free_exec_data()

```
void free_exec_data (
            exec_data_t * exec_data)
```

Frees the memory allocated for the execution data structure.

Cleans up all dynamically allocated memory in the execution data structure, including pipes, process IDs, statuses, and command types.

Parameters

exec_data	Pointer to the execution data structure.

4.5.3.5 get_cmd_path()

```
char * get_cmd_path (
            char * command,
            char ** env)
```

Get the full path of a command.

This function searches for a command in the directories listed in the `PATH` environment variable. If the command is found, its full path is returned. If the command is already an absolute path or relative path, it is returned as is.

Parameters

command	The name of the command to locate.
env	The shell's environment variables.

Returns

A dynamically allocated string containing the full path to the command, or NULL if the command is not found.

4.5.3.6 get_cmd_type()

```
int get_cmd_type (
            commands_t * cmd)
```

Determines the type of a given command.

Parameters

cmd	Pointer to the command structure to analyze.

Returns

An integer representing the type of the command:

- `NOTHING` if the command is NULL or empty.
- `SUBCMD` if the command contains subcommands.
- `BUILT_CMD` if the command is a built-in command.
- `CMD` if the command is an external command.

4.5.3.7 get_paths()

```
char ** get_paths (
            char ** env)
```

Retrieve the list of directories from the `PATH` environment variable.

This function extracts the `PATH` environment variable from the shell's environment and converts it into a list of directories.

Parameters

env	The shell's environment variables.

Returns

A dynamically allocated array of directory paths, or NULL on failure.

4.5.3.8 init_exec_data()

```
int init_exec_data (
            commands_t ** cmds,
            exec_data_t * exec_data)
```

Initializes the execution data structure.

Allocates and initializes the necessary fields in the execution data structure, including pipes, process IDs, statuses, and command types.

Parameters

cmds	Array of commands to execute.
exec_data	Pointer to the execution data structure.

Returns

0 on success, 1 on failure.

4.5.3.9 manage_saved_fd()

```
void manage_saved_fd (
            int saved[2])
```

Restores the saved file descriptors for standard input and output.

Restores the original file descriptors for `STDIN_FILENO` and `STDOUT_FILENO` and closes the saved file descriptors.

Parameters

saved	Array containing the saved file descriptors for stdin and stdout.

4.6 exec.h

Go to the documentation of this file.

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** 42sh
00004 ** File description:
00005 ** exec.h
00006 */
00007
00008 #ifndef EXEC_H
00009     #define EXEC_H
00010
00011     #include "meowsh.h"
00012     #include "builtins.h"
00013     #include <sys/types.h>
00014     #include <sys/wait.h>
00015     #include <fcntl.h>
00016     #include <string.h>
00017     #include <dirent.h>
00018
00019     #define CMD 0
00020     #define BUILT_CMD 1
00021     #define SUBCMD 2
00022     #define NOTHING 3
00023
00024 static const tab_builtins_t tab[NB_BUILT_IN] = {
00025     {"cd", &my_cd, 0},
00026     {"setenv", &my_setenv, 0},
00027     {"env", &my_env, 0},
00028     {"unsetenv", &my_unsetenv, 0},
00029     {"history", &display_history, 0},
00030     {"alias", &interpret_alias, 0},
00031     {"unalias", &interpret_unalias, 0},
00032     {"exit", &my_exit, 0},
00033     {"set", &my_setenv, 1},
00034     {"unset", &my_unsetenv, 1},
00035     {"which", &interpret_which, 0},
00036     {"where", &interpret_where, 0}
00037 };
00038
00039 typedef struct exec_data_s {
00040     int nb_elt;
00041     int **pipe;
00042     int *pid;
00043     int *status;
00044     int *cmd_type;
00045     int prev_depth;
00046 } exec_data_t;
00047
00048 void exec_cmd(shell_t *shell, commands_t **cmds, int prev_depth);
00049 void check_signal(shell_t *shell, exec_data_t *exec_data, int i);
00050 char *get_cmd_path(char *command, char **env);
00051 int get_cmd_type(commands_t *cmd);
00052 char **get_paths(char **env);
00053 int check_path(char *command, char *path);
00054 int init_exec_data(commands_t **cmds, exec_data_t *exec_data);
00055 void free_exec_data(exec_data_t *exec_data);
00056 void manage_saved_fd(int saved[2]);
00057
00058 #endif /*EXEC_H*/
```

4.7 include/format.h File Reference

```
#include "meowsh.h"
#include <stdint.h>
#include <stddef.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
```

Macros

- #define NB_EXCEPTIONS 4
- #define MAX_NBR 11

Functions

- char * add_spaces_around_operators (char *input)

  Adds spaces around operators in a shell input string.
- char * format_input (char *input, shell_t *shell)

  Formats the input string for the shell.
- int handle_skipping (char **array, char **new_array, int i, int *skip_until_separator)

  Handles skipping logic for exceptions and separators.
- int handle_exceptions_and_quotes (char **array, char **new_array, int i, int *skip_after_separator)

  Handles exceptions and quotes for a single argument.
- int check_quote (char **array, char **new_array, int i)

  Checks for quotes in an argument and processes them.

4.7.1 Macro Definition Documentation

4.7.1.1 MAX_NBR

```
#define MAX_NBR 11
```

4.7.1.2 NB_EXCEPTIONS

```
#define NB_EXCEPTIONS 4
```

4.7.2 Function Documentation

4.7.2.1 add_spaces_around_operators()

```
char * add_spaces_around_operators (
            char * input)
```

Adds spaces around operators in a shell input string.

This function processes the input string, identifies operators, and ensures that spaces are added around them. It handles quoted strings to avoid modifying operators inside quotes.

Parameters

input	The input string to process.

Returns

A newly allocated string with spaces added around operators, or NULL on memory allocation failure.

4.7.2.2 check_quote()

```
int check_quote (
            char ** array,
            char ** new_array,
            int i)
```

Checks for quotes in an argument and processes them.

If the argument is enclosed in single quotes, it removes the quotes and returns the processed string.

Parameters

array	The original array of arguments.
new_array	The array to store processed arguments.
i	The index of the current argument.

Returns

0 if successful, 1 on error, or -1 if no quotes are found.

4.7.2.3 format_input()

```
char * format_input (
            char * input,
            shell_t * shell)
```

Formats the input string for the shell.

Splits the input string into arguments, processes each argument for aliases, environment variables, globbing, and history expansion, and concatenates the results into a single formatted string.

Parameters

input	The input string to format.
shell	The shell structure containing aliases, environment variables, and history.

Returns

A newly allocated string containing the formatted input, or NULL on error.

4.7.2.4 handle_exceptions_and_quotes()

```
int handle_exceptions_and_quotes (
            char ** array,
            char ** new_array,
            int i,
            int * skip_after_separator)
```

Handles exceptions and quotes for a single argument.

This function checks if the argument is an exception or contains quotes and processes it accordingly.

Parameters

array	The original array of arguments.
new_array	The array to store processed arguments.
i	The index of the current argument.

Returns

1 on error, 0 if the argument was processed, or -1 if skipped.

4.7.2.5 handle_skipping()

```
int handle_skipping (
            char ** array,
            char ** new_array,
            int i,
            int * skip_until_separator)
```

Handles skipping logic for exceptions and separators.

This function checks if the current argument should be skipped due to an exception or until the next separator or redirection.

Parameters

array	The original array of arguments.
new_array	The array to store formatted arguments.
i	The index of the current argument.
skip_until_separator	Pointer to the skip flag.

Returns

1 if skipping, 0 otherwise.

4.8 format.h

Go to the documentation of this file.

```
00001 /*
00002 ** EPITECH PROJECT, 2025
00003 ** 42sh
00004 ** File description:
00005 ** format header
00006 */
00007
00008 #ifndef FORMAT
00009     #define FORMAT
00010     #include "meowsh.h"
00011     #include <stdint.h>
00012     #include <stddef.h>
00013     #include <stdlib.h>
00014     #include <string.h>
00015     #include <stdio.h>
00016     #define NB_EXCEPTIONS 4
00017     #define MAX_NBR 11
00018
00019 static const char *exceptions[NB_EXCEPTIONS] __attribute__((unused)) = {
```

```
00020     "alias",
00021     "unalias",
00022     "which",
00023     "where"
00024 };
00025
00026 char *add_spaces_around_operators(char *input);
00027 char *format_input(char *input, shell_t *shell);
00028 int handle_skipping(char **array, char **new_array,
00029     int i, int *skip_until_separator);
00030 int handle_exceptions_and_quotes(char **array,
00031     char **new_array, int i, int *skip_after_separator);
00032 int check_quote(char **array, char **new_array, int i);
00033
00034 #endif
```

4.9 include/globbings.h File Reference

```
#include <glob.h>
#include "meowsh.h"
#include <sys/stat.h>
```

Macros

- #define MAX_GLOBS 2048

Functions

- int is_gobbling (char *str)

  Checks if a string contains globbing characters.
- char * get_globbings (char *find)

  Retrieve globbing matches as a concatenated string.
- char * create_concat_array (char **pathv)

  Concatenate an array of strings into a single alloc'ed string.

4.9.1 Macro Definition Documentation

4.9.1.1 MAX_GLOBS

```
#define MAX_GLOBS 2048
```

4.9.2 Function Documentation

4.9.2.1 create_concat_array()

```
char * create_concat_array (
            char ** pathv)
```

Concatenate an array of strings into a single alloc'ed string.

This static function concatenates all strings from the provided array, separating each with a space. Memory is dynamically allocated for the new string and should be freed by the caller.

Parameters

pathv	Array of strings.

Returns

Pointer to the concatenated string, or NULL on failure.

4.9.2.2 get_globbings()

```
char * get_globbings (
            char * find)
```

Retrieve globbing matches as a concatenated string.

This function uses the glob() library function to find matches for the provided pattern. When matches are found, they are concatenated into a single string. The allocated memory must be freed by the caller.

Parameters

find	The globbing pattern.

Returns

Pointer to the concatenated result string, or NULL if no matches are found or on error.

4.9.2.3 is_gobbling()

```
int is_gobbling (
            char * str)
```

Checks if a string contains globbing characters.

This function iterates through the input string and checks for the presence of globbing characters (`*`, `?`, `[`, `]`). If any of these characters are found, the function returns `1`. Otherwise, it returns `0`.

Parameters

str	The input string to check for globbing characters.

Returns

1 if the string contains globbing characters, 0 otherwise.

4.10 globbings.h

Go to the documentation of this file.

```
00001 /*
00002 ** EPITECH PROJECT, 2025
00003 ** 42sh
00004 ** File description:
00005 ** globbings
00006 */
00007
00008 #ifndef GLOBBINGS_H_
00009     #define GLOBBINGS_H_
00010     #include <glob.h>
00011     #include "meowsh.h"
00012     #include <sys/stat.h>
00013     #define MAX_GLOBS 2048
00014
00015 int is_gobbling(char *str);
00016 char *get_globbings(char *find);
00017 char *create_concat_array(char **pathv);
00018
00019 #endif /* !GLOBBINGS_H_ */
```

4.11 include/input.h File Reference

```
#include <termios.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "meowsh.h"
```

Classes

- struct input_s

Macros

- #define BLUE "\033[1;34m"
- #define RESET "\033[0m"
- #define PURPLE "\033[0;38;5;56m"

Typedefs

- typedef struct input_s input_t

Functions

- int handle_sequences (input_t *input, shell_t *shell)
  Handles and redirects all sequences found in the user input.
- int manage_input (input_t *input, shell_t *shell)
  Manages user input by handling special keys and printable characters.
- char * get_input (shell_t *shell)
  Reads and processes user input in raw mode.
- int init_input (input_t **input, shell_t *shell)
  Initializes the input structure.
- void free_input (input_t *input)
  Frees the memory allocated for the input structure.
- int check_shortcut (input_t *input)
  Handles keyboard shortcuts for cursor movement and screen clearing.
- int display_line (input_t *input, shell_t *shell)
  Displays the current input line in the terminal.
- int autocomplete_tabs (input_t *input, shell_t *shell)
  Handles tab-autocompletion by invoking globbing logic.

4.11.1 Macro Definition Documentation

4.11.1.1 BLUE

```
#define BLUE "\033[1;34m"
```

4.11.1.2 PURPLE

```
#define PURPLE "\033[0;38;5;56m"
```

4.11.1.3 RESET

```
#define RESET "\033[0m"
```

4.11.2 Typedef Documentation

4.11.2.1 input_t

```
typedef struct input_s input_t
```

4.11.3 Function Documentation

4.11.3.1 autocomplete_tabs()

```
int autocomplete_tabs (
            input_t * input,
            shell_t * shell)
```

Handles tab-autocompletion by invoking globbing logic.

This function processes the current input to provide tab-autocompletion. It extracts the current word being typed, constructs a globbing pattern, and uses it to find matches. If a single match is found, it replaces the word in the input. If multiple matches are found, it displays them on the terminal. The function also updates the input structure and redisplays the input line after processing.

Parameters

input	Pointer to the input structure containing the current input string, cursor position, and input length.
shell	Pointer to the shell structure containing environment variables and other shell-related data.

Returns

0 on success, 1 on memory allocation failure or other errors.

4.11.3.2 check_shortcut()

```
int check_shortcut (
            input_t * input)
```

Handles keyboard shortcuts for cursor movement and screen clearing.

This function checks for specific keyboard shortcuts:

- `CTRL-A` (ASCII 1): Moves the cursor to the beginning of the line.
- `CTRL-E` (ASCII 5): Moves the cursor to the end of the line.
- `CTRL-B` (ASCII 2): Moves the cursor backward by one character.
- `CTRL-F` (ASCII 6): Moves the cursor forward by one character.
- `CTRL-T` (ASCII 20): Swaps the last two characters in the input string.
- `CTRL-L` (ASCII 12): Clears the terminal screen and resets the cursor.

Parameters

input	Pointer to the input structure containing the current input string, its length, and the cursor position.

Returns

Always returns 0.

4.11.3.3 display_line()

```
int display_line (
            input_t * input,
            shell_t * shell)
```

Displays the current input line in the terminal.

This function clears the current line in the terminal, writes the input string, and repositions the cursor to the correct position.

Parameters

input	Pointer to the input structure containing the current input string, its length, and the cursor position.
shell	The struct containing all the data about the shell

Returns

0 on success.

4.11.3.4 free_input()

```
void free_input (
            input_t * input)
```

Frees the memory allocated for the input structure.

This function frees the memory allocated for the `input_t` structure, including the dynamically allocated input string.

Parameters

input	Pointer to the input structure to free.

4.11.3.5 get_input()

```
char * get_input (
            shell_t * shell)
```

Reads and processes user input in raw mode.

This function reads user input character by character, processes special keys, and updates the input string and cursor position accordingly.

Parameters

shell	Pointer to the shell structure (not used in this function but kept for consistency with the shell's architecture).

Returns

A dynamically allocated string containing the user input, or NULL on error.

4.11.3.6 handle_sequences()

```
int handle_sequences (
            input_t * input,
            shell_t * shell)
```

Handles and redirects all sequences found in the user input.

This function detects if a sequence is found, reads the sequence, and redirects arrow key sequences to another function. It also handles the Suppr (Delete) key sequence directly.

Parameters

input	Pointer to the input structure containing the current input string, its length, and the cursor position.
shell	Pointer to the shell structure.

Returns

1 if a sequence is detected and handled, 0 otherwise.

4.11.3.7 init_input()

```
int init_input (
            input_t ** input,
            shell_t * shell)
```

Initializes the input structure.

This function allocates memory for the `input_t` structure and initializes its fields to default values.

Parameters

input	Double pointer to the input structure to initialize.
shell	Pointer to the shell structure containing environment variables and other shell-related data.

Returns

0 on success, 1 on memory allocation failure.

4.11.3.8 manage_input()

```
int manage_input (
            input_t * input,
            shell_t * shell)
```

Manages user input by handling special keys and printable characters.

This function processes user input character by character. It handles special keys (e.g., Backspace, Delete) and inserts printable characters into the input string. It also updates the input length and cursor position accordingly.

Parameters

input	Pointer to the input structure containing the current input string, its length, and the cursor position.
shell	Pointer to the shell structure containing environment variables and other shell-related data.

Returns

0 on success, 1 on memory allocation failure.

4.12 input.h

Go to the documentation of this file.

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** B-PSU-200-LIL-2-1-42sh-mato.urbanac
00004 ** File description:
00005 ** input.h
00006 */
00007
00008 #ifndef INPUT_H
00009     #define INPUT_H
00010
00011     #include <termios.h>
00012     #include <unistd.h>
00013     #include <stdio.h>
00014     #include <stdlib.h>
00015     #include <string.h>
00016     #include <ctype.h>
00017     #include "meowsh.h"
00018     #define BLUE "\033[1;34m"
00019     #define RESET "\033[0m"
00020     #define PURPLE "\033[0;38;5;56m"
00021
00022 typedef struct input_s {
00023     char *str_input;
00024     char char_input;
00025     size_t input_len;
00026     size_t cursor_pos;
00027     int history_pos;
00028     char *stock_history;
00029 } input_t;
00030
00031 int handle_sequences(input_t *input, shell_t *shell);
00032 int manage_input(input_t *input, shell_t *shell);
00033 char *get_input(shell_t *shell);
00034 int init_input(input_t **input, shell_t *shell);
00035 void free_input(input_t *input);
00036 int check_shortcut(input_t *input);
00037 int display_line(input_t *input, shell_t *shell);
00038 int autocomplete_tabs(input_t *input, shell_t *shell);
00039
00040 #endif
```

4.13 include/meowsh.h File Reference

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <ctype.h>
#include <string.h>
```

Classes

- struct env_s
- struct aliases_s
- struct commands_s
- struct shell_s
- struct tab_builtins_t

Typedefs

- typedef enum sep_s sep_t
- typedef struct env_s env_t
- typedef struct aliases_s aliases_t
- typedef struct commands_s commands_t
- typedef struct shell_s shell_t

Enumerations

- enum sep_s {
  NONE , PIPE , AND , OR ,
  COLUMN , AND_OR }

Functions

- int str_isalphanum (char *str)
  Checks if a string is alphanumeric.

4.13.1 Typedef Documentation

4.13.1.1 aliases_t

```
typedef struct aliases_s aliases_t
```

4.13.1.2 commands_t

```
typedef struct commands_s commands_t
```

4.13.1.3 env_t

```
typedef struct env_s env_t
```

4.13.1.4 sep_t

```
typedef enum sep_s sep_t
```

4.13.1.5 shell_t

```
typedef struct shell_s shell_t
```

4.13.2 Enumeration Type Documentation

4.13.2.1 sep_s

```
enum sep_s
```

Enumerator

NONE	
PIPE	
AND	
OR	
COLUMN	
AND_OR	

4.13.3 Function Documentation

4.13.3.1 str_isalphanum()

```
int str_isalphanum (
            char * str)
```

Checks if a string is alphanumeric.

This function iterates through the input string and checks if all characters are alphanumeric (letters or digits). If any character is not alphanumeric, the function returns `0`. Otherwise, it returns `1`.

Parameters

str	The input string to check.

Returns

1 if the string is alphanumeric, 0 otherwise.

4.14 meowsh.h

Go to the documentation of this file.

```
00001 /*
00002 ** EPITECH PROJECT, 2025
00003 ** meowsh
00004 ** File description:
00005 ** Main header file
00006 */
00007
00008 #ifndef MEOWSH_H
00009     #define MEOWSH_H
00010     #include <unistd.h>
00011     #include <stdlib.h>
00012     #include <stdio.h>
00013     #include <stdbool.h>
00014     #include <ctype.h>
00015     #include <string.h>
00016
00017 typedef enum sep_s {
00018     NONE,
00019     PIPE,
00020     AND,
00021     OR,
00022     COLUMN,
00023     AND_OR
00024 } sep_t;
00025
00027 typedef struct env_s {
00028     char **local;
00029     char **global;
00030 } env_t;
00031
00033
00035 typedef struct aliases_s {
00036     char *original;
00037     char *new_name;
00038     bool browsed;
00039     struct aliases_s *next;
00040 } aliases_t;
00041
00043
00045 typedef struct commands_s {
00046     sep_t separator;
00047     int depth;
00048     int fd_input;
00049     int fd_output;
00050     char *heredoc;
```

```
00051     char *path;
00052     char **command;
00053     struct commands_s **subcommands;
00054 } commands_t;
00055
00056 typedef struct shell_s {
00057     commands_t **commands;
00058     env_t *env;
00059     aliases_t **aliases;
00060     char **history;
00061     int history_size;
00062     char *owd;
00063     int exit_status;
00064     bool run;
00065 } shell_t;
00066
00068
00070 typedef struct {
00071     char *command;
00072     int (*builtin_func)(shell_t *, char **command, int local);
00073     int __attribute_maybe_unused__ local;
00074 } tab_builtins_t;
00075
00077
00078 // Utilities
00079 int str_isalphanum(char *str);
00080
00081 #endif
```

4.15 include/parser.h File Reference

```
#include "meowsh.h"
#include <string.h>
#include <fcntl.h>
#include "utilities.h"
```

Classes

- struct parsing_state_s

Typedefs

- typedef struct parsing_state_s parsing_state_t

Functions

- commands_t * parse_line (char *line)

  Entry point for parsing a command line.
- commands_t * parse_commands (char *line, commands_t *commands, sep_t separator)

  Parse a full command line recursively.
- void change_depth (parsing_state_t *parsing_elements, char current_character)

  Modify the depth counter when encountering parentheses.
- int is_nested (parsing_state_t parsing_elements)

  Check if the parser is currently in a nested state.
- void change_backtick_and_quotes (parsing_state_t *parsing_elements, char current_character)

  Toggle quote or backtick status depending on the character.
- int is_command_inside_parenthesis (char *line)

  Check if the command is entirely wrapped in parentheses.

- char * cut_parenthesis (char *line)

  Remove outer parentheses from the command line.
- commands_t * choose_parsing (char *line, commands_t *commands, sep_t separator, int depth)

  Parses a command line and constructs a command structure based on separators.
- int is_simple_separator (char current_character, sep_t separator)

  Checks if the current character is a separator.
- int is_double_separator (char *line, int *current, sep_t separator, bool does_increment)

  Checks if the current and following character are separators.
- sep_t get_highest_separator (char *line)

  Get the highest-priority separator in the line.
- commands_t * create_command_node (void)

  Initialize a new command node.
- commands_t * handle_no_separator_command (char *line, commands_t *commands)

  Handle parsing when no separator is found.
- int open_output_fd (commands_t *cmd, bool truncate, char *filename)

  Opens the output file descriptor for a command.
- int open_input_fd (commands_t *cmd, char *filename)

  Opens the input file descriptor for a command.
- commands_t * add_command (char *line, commands_t *cmd)

  Add a command to the current commands_t node.
- commands_t * check_input_redirection (char *line, commands_t *cmd, int index)

  Checks for input redirection in the command line and processes it.
- commands_t * check_output_redirection (char *line, commands_t *cmd, int *index)

  Checks for output redirection in the command line and processes it.
- int is_valid_syntax (char *line)

  Checks the line for errors.
- int is_valid_pipe_syntax (char *line)

  Checks if the pipe syntax in the command line is valid.
- bool is_nesting_matched (char *line)

  Checks if quotes and parentheses are matched in the input line.
- bool is_space (char c)

  Checks if a character is a space, tab, or newline.
- int get_number_subcommands (sep_t separator, char *line)

  Get the number of subcommands in a command.
- void update_quote_state (char c, bool *in_single_quote, bool *in_double_quote)

  Updates the quote states depending on the current character.
- bool is_null_command (const char *line, int i)

  Determines if a null command follows the current pipe character.
- int is_valid_and_syntax (char *line)

  Checks if the && syntax in the command line is valid.
- int is_valid_semicolon_syntax (char *line)

  Checks if the ; syntax in the command line is valid.
- sep_t change_separator (sep_t highest_separator, char *line, int i)

  Update the highest separator found in the line.

4.15.1 Typedef Documentation

4.15.1.1 parsing_state_t

```
typedef struct parsing_state_s parsing_state_t
```

4.15.2 Function Documentation

4.15.2.1 add_command()

```
commands_t * add_command (
             char * line,
             commands_t * commands)
```

Add a command to the current commands_t node.

Creates a subcommand and prepares it for parsing.

Parameters

line	The command line to parse.
commands	The parent command node to attach to.

Returns

Pointer to the subcommand created, or NULL on error.

4.15.2.2 change_backtick_and_quotes()

```
void change_backtick_and_quotes (
             parsing_state_t * parsing_elements,
             char current_character)
```

Toggle quote or backtick status depending on the character.

Parameters

parsing_elements	Pointer to the parsing elements' state
current_character	Current character being read.

4.15.2.3 change_depth()

```
void change_depth (
             parsing_state_t * parsing_elements,
             char current_character)
```

Modify the depth counter when encountering parentheses.

Increments or decrements the depth counter depending on the character.

Parameters

parsing_elements	Pointer to the parsing elements' state
current_character	Current character being read.

4.15.2.4 change_separator()

```
sep_t change_separator (
            sep_t highest_separator,
            char * line,
            int i)
```

Update the highest separator found in the line.

Compares the current highest and the one found at position i.

Parameters

highest_separator	The current highest separator.
line	The input command line.
i	Current index being checked.

Returns

The updated highest separator.

4.15.2.5 check_input_redirection()

```
commands_t * check_input_redirection (
             char * line,
             commands_t * commands,
             int i)
```

Checks for input redirection in the command line and processes it.

This function parses a possible input redirection ('<') in the line, handles heredoc redirection ('<<'), and opens the corresponding input file for standard input redirection.

Parameters

line	The command line string being parsed (will be modified).
commands	The structure to store the input file descriptor or heredoc.
i	The current index in the line where the '<' character was found.

Returns

A pointer to the updated commands structure, or NULL on error.

4.15.2.6 check_output_redirection()

```
commands_t * check_output_redirection (
             char * line,
             commands_t * commands,
             int * i)
```

Checks for output redirection in the command line and processes it.

This function handles both simple (>) and append (>>) output redirection. It opens the corresponding output file and stores its file descriptor in `commands->fd_output`.

Parameters

line	The command line string being parsed (will be modified).
commands	The command structure to store the output fd in
i	A pointer to the current index in the line where the '>' was found

Returns

A pointer to the updated commands structure, or NULL on error.

4.15.2.7 choose_parsing()

```
commands_t * choose_parsing (
             char * line,
             commands_t * commands,
             sep_t separator,
             int depth)
```

Parses a command line and constructs a command structure based on separators.

This function analyzes the input line, handles command separators (e.g., `;`, `&&`, `||`), and splits the command into subcommands. It also manages the depth for nested commands (such as parentheses) and assigns the good separator to each subcommand.

Parameters

line	The input command line.
commands	The parent command node.
separator	The separator to use if needed to cut
depth	The current depth, used for parenthesis handling

Returns

A pointer to the new added command, or NULL on error.

4.15.2.8 create_command_node()

```
commands_t * create_command_node (
             void )
```

Initialize a new command node.

Allocates and initializes a new commands_t structure with default values.

Returns

Pointer to the newly created commands_t, or NULL on failure.

4.15.2.9 cut_parenthesis()

```
char * cut_parenthesis (
             char * line)
```

Remove outer parentheses from the command line.

Allocates and returns a copy of the line without outer parentheses.

Parameters

line	The input command line.

Returns

A new string without the outer parentheses, or NULL on failure.

4.15.2.10 get_highest_separator()

```
sep_t get_highest_separator (
            char * line)
```

Get the highest-priority separator in the line.

Skips nested structures like parentheses, quotes and backticks.

Parameters

line	The input command line.

Returns

The most important separator found, or -1 if none.

4.15.2.11 get_number_subcommands()

```
int get_number_subcommands (
            sep_t separator,
            char * line)
```

Get the number of subcommands in a command.

Counts the number of subcommands using a separator.

Parameters

separator	The separator to cut my line.
line	The input command line.

Returns

The number of subcommands.

4.15.2.12 handle_no_separator_command()

```
commands_t * handle_no_separator_command (
            char * line,
            commands_t * commands)
```

Handle parsing when no separator is found.

If the command is nested in parentheses, remove them and parse again. Otherwise, just add the command directly.

Parameters

line	The input command line.
commands	The current commands_t node.

Returns

The updated commands_t after handling the command.

4.15.2.13 is_command_inside_parenthesis()

```
int is_command_inside_parenthesis (
            char * line)
```

Check if the command is entirely wrapped in parentheses.

Parameters

line	The input command line.

Returns

1 if command is parenthesized, 0 otherwise.

4.15.2.14 is_double_separator()

```
int is_double_separator (
            char * line,
            int * current,
            sep_t separator,
            bool does_increment)
```

Checks if the current and following character are separators.

Only in case of AND/OR separator.

Parameters

line	The input command line.
current	The current character index.
separator	The highest sep_t defined separator.
does_increment	true if current has to be increased, else false.

Returns

1 if it's a separator, else 0.

4.15.2.15 is_nested()

```
int is_nested (
            parsing_state_t parsing_elements)
```

Check if the parser is currently in a nested state.

Determines if the parser is inside quotes, backticks or parentheses.

Parameters

parsing_elements	The parsing elements' state

Returns

1 if nested, 0 otherwise.

4.15.2.16 is_nesting_matched()

```
bool is_nesting_matched (
            char * line)
```

Checks if quotes and parentheses are matched in the input line.

Parameters

line	The input string to validate.

Returns

true if all quotes and parentheses are closed, false otherwise.

4.15.2.17 is_null_command()

```
bool is_null_command (
            const char * line,
            int i)
```

Determines if a null command follows the current pipe character.

Skips spaces and checks if there's nothing valid after the pipe.

Parameters

line	The command line string.
i	The index of the pipe character.

Returns

true if nothing follows the pipe, false otherwise.

4.15.2.18 is_simple_separator()

```
int is_simple_separator (
            char current_character,
            sep_t separator)
```

Checks if the current character is a separator.

Only in case of semicolumn and pipe.

Parameters

current_character	The current character to compare.
separator	The highest sep_t defined separator.

Returns

1 if it's a separator, else 0.

4.15.2.19 is_space()

```
bool is_space (
            char c)
```

Checks if a character is a space, tab, or newline.

Parameters

c	The character to check.

Returns

true if the character is a whitespace, false otherwise.

4.15.2.20 is_valid_and_syntax()

```
int is_valid_and_syntax (
            char * line)
```

Checks if the `&&` syntax in the command line is valid.

This function ensures that:

- The line does not start with `&&`.
- `&&` are not followed by invalid or missing commands.
- `&&` checks ignore characters inside quotes.

Parameters

line	The command line string to validate.

Returns

1 if the `&&` syntax is valid, 0 otherwise.

4.15.2.21 is_valid_pipe_syntax()

```
int is_valid_pipe_syntax (
            char * line)
```

Checks if the pipe syntax in the command line is valid.

This function ensures that:

- The line does not start with a pipe.
- Pipes are not followed by invalid or missing commands.
- Pipe checks ignore characters inside quotes.

Parameters

line	The command line string to validate.

Returns

1 if the pipe syntax is valid, 0 otherwise.

4.15.2.22 is_valid_semicolon_syntax()

```
int is_valid_semicolon_syntax (
            char * line)
```

Checks if the `;` syntax in the command line is valid.

This function ensures that:

- The line does not start with `;`.
- `;` are not followed by invalid or missing commands.
- `;` checks ignore characters inside quotes.

Parameters

line	The command line string to validate.

Returns

1 if the `;` syntax is valid, 0 otherwise.

4.15.2.23 is_valid_syntax()

```
int is_valid_syntax (
            char * line)
```

Checks the line for errors.

Validates the syntax.

Parameters

line	The command line to check.

Returns

1 if the line is well formated, else 0.

4.15.2.24 open_input_fd()

```
int open_input_fd (
            commands_t * cmd,
            char * filename)
```

Opens the input file descriptor for a command.

Parameters

cmd	Pointer to the command structure (commands_t).
filename	The path to the file to open as input.

Returns

The new input file descriptor, or -1 on failure.

4.15.2.25 open_output_fd()

```
int open_output_fd (
            commands_t * cmd,
            bool truncate,
            char * filename)
```

Opens the output file descriptor for a command.

Parameters

cmd	Pointer to the command structure (commands_t).
truncate	If true, the file is truncated; if false, data is appended.
filename	The path to the file to open as output.

Returns

The new output file descriptor, or -1 on failure.

4.15.2.26 parse_commands()

```
commands_t * parse_commands (
            char * line,
            commands_t * commands,
            sep_t separator)
```

Parse a full command line recursively.

Splits the line by the highest-level separator and recursively parses subcommands.

Parameters

line	The input command line.
commands	The parent command node.
separator	The separator to use to cut the line

Returns

A pointer to the new added command, or NULL on error.

4.15.2.27 parse_line()

```
commands_t * parse_line (
            char * line)
```

Entry point for parsing a command line.

Validates the syntax and initializes the command tree.

Parameters

line	The command line to parse.

Returns

The root of the parsed command tree, or NULL on error.

4.15.2.28 update_quote_state()

```
void update_quote_state (
            char c,
            bool * in_single_quote,
            bool * in_double_quote)
```

Updates the quote states depending on the current character.

This function toggles the in_single_quote or in_double_quote flags depending on whether the current character is a quote and whether we are currently inside the other type of quote.

Parameters

c	The current character.
in_single_quote	Pointer to a flag : if inside single quotes.
in_double_quote	Pointer to a flag : if inside double quotes.

4.16 parser.h

Go to the documentation of this file.

```
00001 /*
00002 ** EPITECH PROJECT, 2025
00003 ** parser
00004 ** File description:
00005 ** Header file for parsing
00006 */
00007
00008 #ifndef PARSER_H
00009     #define PARSER_H
00010     #include "meowsh.h"
00011     #include <string.h>
00012     #include <fcntl.h>
00013     #include "utilities.h"
00014
00015 // Structs
00016
00017 typedef struct parsing_state_s {
00018     char *line;
00019     sep_t separator;
00020     int current_depth;
00021     bool in_single_quote;
00022     bool in_double_quote;
00023     bool in_backtick;
00024     int start_index;
00025     int current_subcommand_index;
00026 } parsing_state_t;
00027
00028 // main_parsing
00029
00030 commands_t *parse_line(char *line);
00031 commands_t *parse_commands(char *line, commands_t *commands, sep_t separator);
00032 void change_depth(parsing_state_t *parsing_elements, char current_character);
```

```
00033 int is_nested(parsing_state_t parsing_elements);
00034 void change_backtick_and_quotes(parsing_state_t *parsing_elements,
00035     char current_character);
00036 int is_command_inside_parenthesis(char *line);
00037 char *cut_parenthesis(char *line);
00038 commands_t *choose_parsing(char *line, commands_t *commands,
00039     sep_t separator, int depth);
00040 // command_handling
00041
00042 int is_simple_separator(char current_character, sep_t separator);
00043 int is_double_separator(char *line, int *current,
00044     sep_t separator, bool does_increment);
00045 sep_t get_highest_separator(char *line);
00046 commands_t *create_command_node(void);
00047 commands_t *handle_no_separator_command(char *line, commands_t *commands);
00048 int open_output_fd(commands_t *cmd, bool truncate, char *filename);
00049 int open_input_fd(commands_t *cmd, char *filename);
00050 commands_t *add_command(char *line, commands_t *cmd);
00051 commands_t *check_input_redirection(char *line, commands_t *cmd, int index);
00052 commands_t *check_output_redirection(char *line, commands_t *cmd, int *index);
00053
00054 // is_command_valid
00055
00056 int is_valid_syntax(char *line);
00057 int is_valid_pipe_syntax(char *line);
00058 bool is_nesting_matched(char *line);
00059 bool is_space(char c);
00060 bool is_nesting_matched(char *line);
00061 int get_number_subcommands(sep_t separator, char *line);
00062
00063 void update_quote_state(char c,
00064     bool *in_single_quote, bool *in_double_quote);
00065 bool is_null_command(const char *line, int i);
00066 int is_valid_and_syntax(char *line);
00067 int is_valid_semicolon_syntax(char *line);
00068
00069 sep_t change_separator(sep_t highest_separator, char *line, int i);
00070
00071 #endif
```

4.17 include/utilities.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "meowsh.h"
```

Functions

- int del_elt (char ***array, int index)

  Deletes an element from an array of strings.
- char ** dup_array (char **src)

  Duplicates an array of strings.
- void free_array (char **array)

  Frees a dynamically allocated array of strings.
- void free_shell (shell_t *shell)

  Frees all memory allocated for the shell structure.
- shell_t * init_struct (shell_t *shell, char **envp)

  Initializes a shell structure.
- void free_commands (commands_t **cmds)

  Frees memory allocated for an array of commands.
- char ** my_str_to_word_array (const char *str, const char *spliters)

  Splits a string into an array of words.
- char * strjoin (char *str1, char *str2)

  Concatenate two strings into a new string.
- char * int_to_str (int value)

  Converts a non-negative integer to a dynamically allocated string.

4.17.1 Function Documentation

4.17.1.1 del_elt()

```
int del_elt (
            char *** array,
            int index)
```

Deletes an element from an array of strings.

This function removes the element at the specified index from the array and shifts the remaining elements to fill the gap. The array is resized accordingly.

Parameters

array	Pointer to the array of strings.
index	The index of the element to delete.

Returns

0 on success, 1 on failure (e.g., invalid index or memory allocation failure).

4.17.1.2 dup_array()

```
char ** dup_array (
            char ** src)
```

Duplicates an array of strings.

This function creates a new array of strings that is a deep copy of the source array. Each string in the source array is duplicated.

Parameters

src	The source array of strings to duplicate.

Returns

A newly allocated array of strings, or NULL on failure.

4.17.1.3 free_array()

```
void free_array (
            char ** array)
```

Frees a dynamically allocated array of strings.

This function iterates through the array, freeing each string, and then frees the array itself.

Parameters

array	The array of strings to free.

4.17.1.4 free_commands()

```
void free_commands (
            commands_t ** cmds)
```

Frees memory allocated for an array of commands.

This function iterates through an array of `commands_t` structures and frees all associated memory, including file descriptors, heredoc strings, paths, commands, and subcommands.

Parameters

cmds	Double pointer to the array of commands to free.

4.17.1.5 free_shell()

```
void free_shell (
            shell_t * shell)
```

Frees all memory allocated for the shell structure.

This function frees all dynamically allocated memory associated with the `shell_t` structure, including commands, environment variables, aliases, history, and the old working directory.

Parameters

shell	Pointer to the shell structure to free.

4.17.1.6 init_struct()

```
shell_t * init_struct (
            shell_t * shell,
            char ** envp)
```

Initializes a shell structure.

This function allocates memory for a shell_t structure and its associated components, including commands, environment variables, aliases, and history. It also initializes the shell's working directory and exit status.

Parameters

shell	A pointer to a shell_t structure. If NULL, a new structure will be allocated and initialized.
envp	An array of environment variable strings. This will be used to populate the global environment variables in the shell.

Returns

A pointer to the initialized shell_t structure on success, or NULL if memory allocation fails or if any of the components cannot be initialized.

Note

The caller is responsible for freeing the allocated memory for the shell_t structure and its components when they are no longer needed.

4.17.1.7 int_to_str()

```
char * int_to_str (
            int value)
```

Converts a non-negative integer to a dynamically allocated string.

Allocates memory to store the string representation of the given integer and formats it into the allocated memory.

Parameters

value	The integer value to convert (must be non-negative).

Returns

A dynamically allocated string containing the number representation, or NULL on allocation failure or invalid input.

4.17.1.8 my_str_to_word_array()

```
char ** my_str_to_word_array (
            const char * str,
            const char * spliters)
```

Splits a string into an array of words.

This function splits the input string into an array of words based on the specified delimiters. Quoted sections are treated as a single word.

Parameters

str	The input string to split.
spliters	A string containing the delimiter characters.

Returns

A dynamically allocated array of strings, or NULL on failure.

4.17.1.9 strjoin()

```
char * strjoin (
            char * str1,
            char * str2)
```

Concatenate two strings into a new string.

This function takes two input strings, allocates memory for a new string that contains the concatenation of the two, and returns the new string.

Parameters

str1	The first string to concatenate.
str2	The second string to concatenate.

Returns

A dynamically allocated string containing the concatenation of `str1` and `str2`, or `NULL` if memory allocation fails.

4.18 utilities.h

Go to the documentation of this file.

```
00001 /*
00002 ** EPITECH PROJECT, 2025
00003 ** 42sh
00004 ** File description:
00005 ** utilities header
00006 */
00007
00008 #ifndef UTILITIES
00009     #define UTILITIES
00010
00011     #include <stdio.h>
00012     #include <stdlib.h>
00013     #include <string.h>
00014     #include "meowsh.h"
00015
00016 int del_elt(char ***array, int index);
00017 char **dup_array(char **src);
00018 void free_array(char **array);
00019 void free_shell(shell_t *shell);
00020 shell_t *init_struct(shell_t *shell, char **envp);
00021 void free_commands(commands_t **cmds);
00022 char **my_str_to_word_array(const char *str, const char *spliters);
00023 char *strjoin(char *str1, char *str2);
00024 char *int_to_str(int value);
00025
00026 #endif
```

4.19 src/builtins/alias/alias.c File Reference

All functions to handle the adding of aliases.

```
#include "builtins.h"
#include "globbings.h"
```

Functions

- int interpret_alias (shell_t *shell, char **command, __attribute_maybe_unused__ int local)

  Handles the `alias` command in the shell.

4.19.1 Detailed Description

All functions to handle the adding of aliases.

This file contains all the functions to add an alias

4.19.2 Function Documentation

4.19.2.1 interpret_alias()

```
int interpret_alias (
            shell_t * shell,
            char ** command,
            __attribute_maybe_unused__ int local)
```

Handles the `alias` command in the shell.

This function interprets the `alias` command based on the number of arguments provided:

- If no arguments are provided, it displays all existing aliases.
- If one argument is provided, it does nothing (reserved for future use).
- If two or more arguments are provided, it adds a new alias to the list.

Parameters

shell	The shell structure containing the alias list.
command	The array of arguments passed to the `alias` command.
local	Unused parameter (reserved for future use).

Returns

0 on success, 1 on failure.

4.20 src/builtins/alias/handle_alias.c File Reference

Functions to manage aliases in the shell.

```
#include "builtins.h"
```

Functions

- void add_node (aliases_t **aliases, aliases_t *new_node)
  Add a new alias to the aliases linked list.
- void free_node (aliases_t *node)
  Free memory allocated for an alias node.
- void delete_node (aliases_t **aliases, aliases_t *current, aliases_t *previous)
  Delete an alias from the aliases linked list.

4.20.1 Detailed Description

Functions to manage aliases in the shell.

This file contains functions to handle the creation, addition, and deletion of aliases in the shell. Aliases allow users to define shortcuts for commands.

4.20.2 Function Documentation

4.20.2.1 add_node()

```
void add_node (
            aliases_t ** aliases,
            aliases_t * new_node)
```

Add a new alias to the aliases linked list.

This function adds a new alias node to the linked list of aliases. The list is maintained in lexicographical order based on the alias name.

Parameters

aliases	Pointer to the head of the aliases linked list.
new_node	Pointer to the new alias node to add.

4.20.2.2 delete_node()

```
void delete_node (
            aliases_t ** aliases,
            aliases_t * current,
            aliases_t * previous)
```

Delete an alias from the aliases linked list.

This function removes an alias node from the linked list of aliases and frees its memory. If the alias to delete is the head of the list, the head pointer is updated.

Parameters

aliases	Pointer to the head of the aliases linked list.
current	Pointer to the alias node to delete.
previous	Pointer to the alias node preceding the one to delete.

4.20.2.3 free_node()

```
void free_node (
            aliases_t * node)
```

Free memory allocated for an alias node.

This function frees the memory allocated for a single alias node, including its `original` and `new_name` strings.

Parameters

node	Pointer to the alias node to free.

4.21 src/builtins/alias/unalias.c File Reference

Functions to handle the deletion of aliases in the shell.

```
#include "builtins.h"
```

Functions

- int interpret_unalias (shell_t *shell, char **command, __attribute_maybe_unused__ int local)

  Interpret the `unalias` command.

4.21.1 Detailed Description

Functions to handle the deletion of aliases in the shell.

This file contains all the functions required to delete aliases from the shell's alias list. It includes functionality to browse and remove specific aliases based on user input.

4.21.2 Function Documentation

4.21.2.1 interpret_unalias()

```
int interpret_unalias (
            shell_t * shell,
            char ** command,
            __attribute_maybe_unused__ int local)
```

Interpret the `unalias` command.

This function processes the `unalias` command entered by the user. It checks if the command has enough arguments and deletes the specified aliases from the shell's alias list. If no arguments are provided, an error message is displayed.

Parameters

shell	Pointer to the shell structure containing the alias list.
command	Array of strings representing the `unalias` command and its arguments.
local	Unused parameter for compatibility with other builtins.

Returns

Exit code: 0 on success, 1 on failure (e.g., too few arguments).

4.22 src/builtins/cd/cd_tilde.c File Reference

Functions to handle the tilde (~) for the `cd` command.

```
#include "builtins.h"
```

Functions

- char * handle_tilde (char *directory, char **env, char *previous_dir)
  Handles tilde (~) and other special cases for the `cd` command.

4.22.1 Detailed Description

Functions to handle the tilde (~) for the `cd` command.

This file contains all the necessary functions to expand and handle the tilde (~) in the `cd` command.

4.22.2 Function Documentation

4.22.2.1 handle_tilde()

```
char * handle_tilde (
            char * directory,
            char ** env,
            char * previous_dir)
```

Handles tilde (~) and other special cases for the `cd` command.

This function processes the input directory string and handles:

- ~ for the current user's home directory.
- ~`username` for another user's home directory.

Parameters

directory	The input directory string.
env	The environment variables array.
previous_dir	The previous directory string.

Returns

A newly allocated string containing the processed directory.

4.23 src/builtins/cd/get_home_user.c File Reference

Functions to retrieve a user's home directory.

```
#include "builtins.h"
```

Functions

- char * get_root_home (char *user)
  Retrieves the home directory of a specified user.

4.23.1 Detailed Description

Functions to retrieve a user's home directory.

This file contains functions to parse the `/etc/passwd` file and extract the home directory of a specified user.

4.23.2 Function Documentation

4.23.2.1 get_root_home()

```
char * get_root_home (
            char * user)
```

Retrieves the home directory of a specified user.

This function reads the `/etc/passwd` file, searches for the specified user, and extracts their home directory from the corresponding line.

Parameters

user	The username whose home directory is to be retrieved.

Returns

A dynamically allocated string containing the user's home directory, or NULL if the user is not found or an error occurs.

4.24 src/builtins/cd/my_cd.c File Reference

Implementation of the `cd` command for the shell.

```
#include "builtins.h"
```

Functions

- int my_cd (shell_t *shell, char **command_args, __attribute_maybe_unused__ int local)

  Handles the `cd` command.

4.24.1 Detailed Description

Implementation of the `cd` command for the shell.

This file contains functions to handle the `cd` command, including directory switching, error handling, and tilde expansion.

4.24.2 Function Documentation

4.24.2.1 my_cd()

```
int my_cd (
            shell_t * shell,
            char ** command_args,
            __attribute_maybe_unused__ int local)
```

Handles the `cd` command.

This function processes the `cd` command, handling arguments, tilde expansion, and error cases such as too many arguments or invalid paths. It updates the shell's old working directory (`owd`) and switches to the specified directory.

Parameters

shell	The shell structure containing environment variables and the old working directory.
command_args	The arguments passed to the `cd` command.
local	Unused parameter for compatibility with other builtins.

Returns

0 on success, 1 on failure.

4.25 src/builtins/cd/my_getenv.c File Reference

Retrieves the value of an environment variable.

```
#include "builtins.h"
```

Functions

- char * get_value_key (char *key, char **env)

  Retrieves the value associated with a given key in the environment.

4.25.1 Detailed Description

Retrieves the value of an environment variable.

This file contains the implementation of a function to search for a specific key in the environment variables and return its value.

4.25.2 Function Documentation

4.25.2.1 get_value_key()

```
char * get_value_key (
            char * key,
            char ** env)
```

Retrieves the value associated with a given key in the environment.

This function iterates through the environment variables to find a key that matches the specified key and returns its value.

Parameters

key	The key to search for in the environment variables.
env	The array of environment variables.

Returns

A pointer to the value associated with the key, or NULL if the key is not found.

4.26 src/builtins/env/env.c File Reference

Contains the implementation of the `env` command.

```
#include "meowsh.h"
```

Functions

- int my_env (shell_t *sh, char **command, int local)

  Display the environment variables.

4.26.1 Detailed Description

Contains the implementation of the `env` command.

This file provides the function to display the environment variables stored in the shell structure. It supports both global and local environments and handles error cases such as invalid arguments.

4.26.2 Function Documentation

4.26.2.1 my_env()

```
int my_env (
            shell_t * sh,
            char ** command,
            int local)
```

Display the environment variables.

This function prints the environment variables stored in the shell's global or local environment, depending on the `local` parameter. If additional arguments are passed to the `env` command, an error message is displayed, and the shell's exit status is set to `127`.

Parameters

sh	Pointer to the shell structure containing environment variables.
command	Array of strings representing the `env` command and its arguments.
local	If set to `1`, displays the local environment; otherwise, displays the global environment.

Returns

Always returns `0`.

4.27 src/builtins/env/setenv.c File Reference

Implementation of the `setenv` builtin command.

```
#include "builtins.h"
```

Functions

- int is_a_good_var (char *var)

  Checks if a variable name is valid.
- char * get_line_env (char **command)

  Constructs an environment variable assignment string.
- int add_env (char ***env, char *set)

  Adds a new environment variable to the environment.
- int change_env (char ***env, char **command)

  Modifies an existing environment variable or adds a new one.
- int my_setenv (shell_t *shell, char **command, int local)

  Handles the `setenv` builtin command.

4.27.1 Detailed Description

Implementation of the `setenv` builtin command.

This file contains all the functions required to handle the `setenv` command, which is used to set or modify environment variables in the shell.

4.27.2 Function Documentation

4.27.2.1 add_env()

```
int add_env (
             char *** env,
             char * set)
```

Adds a new environment variable to the environment.

This function appends a new variable to the environment array.

Parameters

env	A pointer to the environment array.
set	The variable assignment string to add (e.g., `VAR=VALUE`).

Returns

0 on success, 1 on failure.

4.27.2.2 change_env()

```
int change_env (
              char *** env,
              char ** command)
```

Modifies an existing environment variable or adds a new one.

This function searches for an existing variable in the environment and updates its value. If the variable does not exist, it adds a new one.

Parameters

env	A pointer to the environment array.
command	The command arguments, where `command[1]` is the variable name and `command[2]` is the value (optional).

Returns

0 on success, 1 on failure.

4.27.2.3 get_line_env()

```
char * get_line_env (
            char ** command)
```

Constructs an environment variable assignment string.

This function creates a string in the format `VAR=VALUE` based on the provided command arguments.

Parameters

command	The command arguments, where `command[1]` is the variable name and `command[2]` is the value (optional).

Returns

A dynamically allocated string containing the assignment, or NULL if memory allocation fails.

4.27.2.4 is_a_good_var()

```
int is_a_good_var (
            char * var)
```

Checks if a variable name is valid.

This function verifies that the variable name starts with a letter or an underscore and contains only alphanumeric characters, underscores, or dots.

Parameters

var	The variable name to check.

Returns

1 if the variable name is valid, 0 otherwise.

4.27.2.5 my_setenv()

```
int my_setenv (
            shell_t * shell,
            char ** command,
            int local)
```

Handles the `setenv` builtin command.

This function implements the logic for the `setenv` command, which sets or modifies environment variables. It validates the input, checks for errors, and updates the appropriate environment (local or global).

Parameters

shell	The shell structure containing the environment.
command	The command arguments, where `command[1]` is the variable name and `command[2]` is the value (optional).
local	A flag indicating whether to modify the local or global environment.

Returns

The correct exit status.

4.28 src/builtins/env/unsetenv.c File Reference

Functions to handle the `unsetenv` builtin command.

```
#include "builtins.h"
```

Functions

- int my_unsetenv (shell_t *sh, char **command, int local)
  Unsets environment variables.
- int add_elt (char ***array, char *elt)
  Adds a new element to the environment array.

4.28.1 Detailed Description

Functions to handle the `unsetenv` builtin command.

This file contains all the functions needed to execute the `unsetenv` builtin command, which removes environment variables from the shell's environment.

4.28.2 Function Documentation

4.28.2.1 add_elt()

```
int add_elt (
              char *** array,
              char * elt)
```

Adds a new element to the environment array.

This function appends a new environment variable to the given environment array.

Parameters

array	Pointer to the environment array.
elt	The environment variable to add.

Returns

int Returns 0 on success, or 1 on failure.

4.28.2.2 my_unsetenv()

```
int my_unsetenv (
            shell_t * sh,
            char ** command,
            int local)
```

Unsets environment variables.

This function implements the `unsetenv` builtin. It removes each environment variable provided in the command arguments. If no variable is given, it writes an error message and sets the shell's exit status.

Parameters

sh	Pointer to the shell structure.
command	Array of command arguments.
local	Non-zero for the local environment, zero for global.

Returns

int Returns 1 if an error occurred, otherwise 0.

4.29 src/builtins/exec/my_exit.c File Reference

Implementation of the `exit` builtin command.

```
#include "builtins.h"
```

Functions

- int my_exit (shell_t *shell, char **command, __attribute_maybe_unused__ int local)

  Handles the `exit` command.

4.29.1 Detailed Description

Implementation of the `exit` builtin command.

This file contains the implementation of the `exit` command, which terminates the shell and optionally sets an exit code.

4.29.2 Function Documentation

4.29.2.1 my_exit()

```
int my_exit (
            shell_t * shell,
            char ** command,
            __attribute_maybe_unused__ int local)
```

Handles the `exit` command.

This function processes the `exit` command, which terminates the shell. If an exit code is provided as an argument, it is used as the shell's exit code. Otherwise, the shell's current `exit_status` is used.

Parameters

shell	Pointer to the shell structure.
command	Array of strings representing the `exit` command and its arguments.
local	Unused parameter for compatibility with other builtins.

Returns

The exit code to terminate the shell with.

4.30 src/builtins/history/handle_history.c File Reference

Functions to manage the shell's command history.

```
#include "builtins.h"
#include "exec.h"
```

Functions

- void free_history (char **history, int count)

  Free a partially filled history.
- void get_history (shell_t *shell)

  Load the history into the shell.
- void add_to_history (char *command, shell_t *shell)

  Add a command to the history and save it to the file.
- int display_history (shell_t *shell, __attribute_maybe_unused__ char **command, __attribute_maybe_↩ unused__ int local)

  Display the entire history.
- char * exec_history (shell_t *shell, char *command)

  Execute a command from the history.

4.30.1 Detailed Description

Functions to manage the shell's command history.

This file contains all the functions required to handle the shell's history, including adding commands, displaying the history, and executing commands from the history.

4.30.2 Function Documentation

4.30.2.1 add_to_history()

```
void add_to_history (
            char * command,
            shell_t * shell)
```

Add a command to the history and save it to the file.

This function adds a command to the shell's history array and appends it to the `.42sh_history` file.

Parameters

command	The command to add to the history.
shell	The shell structure containing the history.

4.30.2.2 display_history()

```
int display_history (
            shell_t * shell,
            __attribute_maybe_unused__ char ** command,
            __attribute_maybe_unused__ int local)
```

Display the entire history.

This function prints the entire history with an index for each command.

Parameters

shell	The shell structure containing the history.
command	Unused parameter.
local	Unused parameter.

Returns

Always returns 0.

4.30.2.3 exec_history()

```
char * exec_history (
            shell_t * shell,
            char * command)
```

Execute a command from the history.

This function searches the history for a command matching the given input and returns it for execution.

Parameters

shell	The shell structure containing the history.
command	The input string specifying the command to execute.

Returns

A string containing the command from the history, or NULL if not found.

4.30.2.4 free_history()

```
void free_history (
            char ** history,
            int count)
```

Free a partially filled history.

This function frees the memory allocated for a partially filled history array, up to the specified count.

Parameters

history	The history array.
count	Number of valid entries to free.

4.30.2.5 get_history()

```
void get_history (
            shell_t * shell)
```

Load the history into the shell.

This function loads the history from the `.42sh_history` file into the shell's history array.

Parameters

shell	The shell structure containing the history.

4.31 src/builtins/which/which.c File Reference

Implementation of the `which` builtin command.

```
#include "builtins.h"
#include "utilities.h"
#include "exec.h"
```

Functions

- int interpret_which (shell_t *shell, char **command, __attribute_maybe_unused__ int local)

  Handle the `which` command.

4.31.1 Detailed Description

Implementation of the `which` builtin command.

This file contains the implementation of the `which` command, which identifies the location of executables or aliases in the shell.

4.31.2 Function Documentation

4.31.2.1 interpret_which()

```
int interpret_which (
            shell_t * shell,
            char ** command,
            __attribute_maybe_unused__ int local)
```

Handle the `which` command.

This function processes the `which` command, which identifies the location of executables or aliases. It checks if the provided arguments are aliases or commands in the `PATH` and prints their locations.

Parameters

shell	Pointer to the shell structure containing aliases and environment variables.
command	Array of strings representing the `which` command and its arguments.
local	Unused parameter for compatibility with other builtins.

Returns

0 on success, 1 on failure (e.g., too few arguments).

4.32 src/builtins/which_and_where/which.c File Reference

Implementation of the `which` builtin command.

```
#include "builtins.h"
#include "utilities.h"
#include "exec.h"
```

Functions

- int is_a_builtin_command_which (char *command)

  Check if a command is a shell built-in for which.
- int interpret_which (shell_t *shell, char **command, __attribute_maybe_unused__ int local)

  Handle the `which` command.

4.32.1 Detailed Description

Implementation of the `which` builtin command.

This file contains the implementation of the `which` command, which identifies the location of executables or aliases in the shell.

4.32.2 Function Documentation

4.32.2.1 interpret_which()

```
int interpret_which (
            shell_t * shell,
            char ** command,
            __attribute_maybe_unused__ int local)
```

Handle the `which` command.

This function processes the `which` command, which identifies the location of executables or aliases. It checks if the provided arguments are aliases or commands in the `PATH` and prints their locations.

Parameters

shell	Pointer to the shell structure containing aliases and environment variables.
command	Array of strings representing the `which` command and its arguments.
local	Unused parameter for compatibility with other builtins.

Returns

0 on success, 1 on failure (e.g., too few arguments).

4.32.2.2 is_a_builtin_command_which()

```
int is_a_builtin_command_which (
            char * command)
```

Check if a command is a shell built-in for which.

This function loops through the `tab` of built-in commands and checks if the given command matches any of the built-ins. If it is a built-in, it prints a message indicating that it is a shell built-in command.

Parameters

command	The command to check.

Returns

1 if the command is a built-in, 0 otherwise.

4.33 src/builtins/which_and_where/where.c File Reference

Implementation of the `where` builtin command.

```
#include "builtins.h"
#include "utilities.h"
#include "exec.h"
```

Functions

- int is_a_builtin_command_where (char *command)

  Check if a command is a shell built-in for where.
- int interpret_where (shell_t *shell, char **command, __attribute_maybe_unused__ int local)

  Handle the `where` command.

4.33.1 Detailed Description

Implementation of the `where` builtin command.

This file contains the implementation of the `where` command, which identifies all the location of executables or aliases in the shell.

4.33.2 Function Documentation

4.33.2.1 interpret_where()

```
int interpret_where (
            shell_t * shell,
            char ** command,
            __attribute_maybe_unused__ int local)
```

Handle the `where` command.

This function processes the `where` command, which identifies all locations of executables or aliases. It checks if the provided arguments are aliases or commands in the `PATH` and prints all their locations.

Parameters

shell	Pointer to the shell structure containing aliases and environment variables.
command	Array of strings representing the `where` command and its arguments.
local	Unused parameter for compatibility with other builtins.

Returns

0 on success, 1 on failure (e.g., too few arguments).

4.33.2.2 is_a_builtin_command_where()

```
int is_a_builtin_command_where (
            char * command)
```

Check if a command is a shell built-in for where.

This function loops through the `tab` of built-in commands and checks if the given command matches any of the built-ins. If it is a built-in, it prints a message indicating that it is a shell built-in command.

Parameters

command	The command to check.

Returns

1 if the command is a built-in, 0 otherwise.

4.34 src/exec/check_signal.c File Reference

Functions to handle and display process signals.

```
#include "exec.h"
```

Functions

- void check_signal (shell_t *shell, exec_data_t *exec_data, int i)

  Updates the shell's exit status based on the process termination status.

4.34.1 Detailed Description

Functions to handle and display process signals.

4.34.2 Function Documentation

4.34.2.1 check_signal()

```
void check_signal (
            shell_t * shell,
            exec_data_t * exec_data,
            int i)
```

Updates the shell's exit status based on the process termination status.

If the process was a built-in command, it directly sets the shell's exit status to the process's exit code. Otherwise, it updates the exit status based on whether the process exited normally or was terminated by a signal.

Parameters

shell	The shell structure containing the exit status.
status	The status of the terminated process.
is_built_cmd	A boolean indicating if the process was a built-in command.

4.35 src/exec/exec_cmd.c File Reference

Implementation of command execution for the 42sh shell.

```
#include "builtins.h"
#include "exec.h"
#include "error_outputs.h"
#include "input.h"
```

Functions

- void exec_cmd (shell_t *shell, commands_t **cmds, int prev_depth)

  Executes a sequence of commands, handling pipes and separators.

4.35.1 Detailed Description

Implementation of command execution for the 42sh shell.

This file contains functions to handle the execution of commands, including handling pipes, redirections, and built-in commands.

4.35.2 Function Documentation

4.35.2.1 exec_cmd()

```
void exec_cmd (
            shell_t * shell,
            commands_t ** cmds,
            int prev_depth)
```

Executes a sequence of commands, handling pipes and separators.

Initializes execution data, processes each command, and cleans up resources.

Parameters

shell	Pointer to the shell structure.
cmds	Array of commands to execute.
prev_depth	Depth of the previous command in the command tree.

4.36 src/exec/get_cmd_path.c File Reference

Functions to locate the executable path of a command.

```
#include "exec.h"
#include "utilities.h"
```

Functions

- char ** get_paths (char **env)
  Retrieve the list of directories from the PATH environment variable.
- int check_path (char *command, char *path)
  Check if a command exists in a given directory.
- char * get_cmd_path (char *command, char **env)
  Get the full path of a command.

4.36.1 Detailed Description

Functions to locate the executable path of a command.

This file contains functions to parse the PATH environment variable, search for a command in the directories listed in PATH, and return the full path to the executable if found.

4.36.2 Function Documentation

4.36.2.1 check_path()

```
int check_path (
            char * command,
            char * path)
```

Check if a command exists in a given directory.

This function searches for a command in the specified directory by iterating through the directory's contents.

Parameters

command	The name of the command to search for.
path	The directory path to search in.

Returns

1 if the command is found, 0 otherwise.

4.36.2.2 get_cmd_path()

```
char * get_cmd_path (
            char * command,
            char ** env)
```

Get the full path of a command.

This function searches for a command in the directories listed in the `PATH` environment variable. If the command is found, its full path is returned. If the command is already an absolute path or relative path, it is returned as is.

Parameters

command	The name of the command to locate.
env	The shell's environment variables.

Returns

A dynamically allocated string containing the full path to the command, or NULL if the command is not found.

4.36.2.3 get_paths()

```
char ** get_paths (
            char ** env)
```

Retrieve the list of directories from the `PATH` environment variable.

This function extracts the `PATH` environment variable from the shell's environment and converts it into a list of directories.

Parameters

env	The shell's environment variables.

Returns

A dynamically allocated array of directory paths, or NULL on failure.

4.37 src/exec/get_cmd_type.c File Reference

Determines the type of a command in the 42sh shell.

```
#include "exec.h"
```

Functions

- int get_cmd_type (commands_t *cmd)
  Determines the type of a given command.

4.37.1 Detailed Description

Determines the type of a command in the 42sh shell.

This file contains the implementation of the `get_cmd_type` function, which identifies whether a command is a built-in command, a subcommand, or an external command. It also handles cases where no command is present.

4.37.2 Function Documentation

4.37.2.1 get_cmd_type()

```
int get_cmd_type (
            commands_t * cmd)
```

Determines the type of a given command.

Parameters

cmd	Pointer to the command structure to analyze.

Returns

An integer representing the type of the command:

- `NOTHING` if the command is NULL or empty.
- `SUBCMD` if the command contains subcommands.
- `BUILT_CMD` if the command is a built-in command.
- `CMD` if the command is an external command.

4.38 src/exec/is_built_command.c File Reference

```
#include "exec.h"
```

Functions

- int is_built_command (char *cmd)

  Determines if a command is a built-in command.

4.38.1 Function Documentation

4.38.1.1 is_built_command()

```
int is_built_command (
            char * cmd)
```

Determines if a command is a built-in command.

This function checks if the given command matches any of the built-in commands defined in the shell. If the command is a built-in, it returns 1. If the command is NULL, it returns 2. Otherwise, it returns 0.

Parameters

cmd	The command to check.

Returns

int

- 1 if the command is a built-in command.
- 2 if the command is NULL.
- 0 if the command is not a built-in command.

4.39 src/exec/manage_exec_data.c File Reference

Manages the initialization, cleanup, and handling of execution data.

```
#include "exec.h"
```

Functions

- int init_exec_data (commands_t **cmds, exec_data_t *exec_data)

  Initializes the execution data structure.
- void free_exec_data (exec_data_t *exec_data)

  Frees the memory allocated for the execution data structure.
- void manage_saved_fd (int saved[2])

  Restores the saved file descriptors for standard input and output.

4.39.1 Detailed Description

Manages the initialization, cleanup, and handling of execution data.

This file contains functions to initialize and free the execution data structure, manage pipes, and restore saved file descriptors.

4.39.2 Function Documentation

4.39.2.1 free_exec_data()

```
void free_exec_data (
            exec_data_t * exec_data)
```

Frees the memory allocated for the execution data structure.

Cleans up all dynamically allocated memory in the execution data structure, including pipes, process IDs, statuses, and command types.

Parameters

exec_data	Pointer to the execution data structure.

4.39.2.2 init_exec_data()

```
int init_exec_data (
            commands_t ** cmds,
            exec_data_t * exec_data)
```

Initializes the execution data structure.

Allocates and initializes the necessary fields in the execution data structure, including pipes, process IDs, statuses, and command types.

Parameters

cmds	Array of commands to execute.
exec_data	Pointer to the execution data structure.

Returns

0 on success, 1 on failure.

4.39.2.3 manage_saved_fd()

```
void manage_saved_fd (
            int saved[2])
```

Restores the saved file descriptors for standard input and output.

Restores the original file descriptors for `STDIN_FILENO` and `STDOUT_FILENO` and closes the saved file descriptors.

Parameters

saved	Array containing the saved file descriptors for stdin and stdout.

4.40 src/format/check_exception.c File Reference

```
#include "meowsh.h"
#include "format.h"
```

Functions

- int is_an_exception (char **array, char **new_array, int i, int *skip_after_sep)

  Checks for exceptions in an argument and ignores them.
- int handle_exceptions_and_quotes (char **array, char **new_array, int i, int *skip_after_separator)

  Handles exceptions and quotes for a single argument.
- int is_separator_or_redirection (const char *arg)

  Checks if the argument is a separator or redirection.
- int handle_skipping (char **array, char **new_array, int i, int *skip_until_separator)

  Handles skipping logic for exceptions and separators.

4.40.1 Function Documentation

4.40.1.1 handle_exceptions_and_quotes()

```
int handle_exceptions_and_quotes (
            char ** array,
            char ** new_array,
            int i,
            int * skip_after_separator)
```

Handles exceptions and quotes for a single argument.

This function checks if the argument is an exception or contains quotes and processes it accordingly.

Parameters

array	The original array of arguments.
new_array	The array to store processed arguments.
i	The index of the current argument.

Returns

1 on error, 0 if the argument was processed, or -1 if skipped.

4.40.1.2 handle_skipping()

```
int handle_skipping (
            char ** array,
            char ** new_array,
            int i,
            int * skip_until_separator)
```

Handles skipping logic for exceptions and separators.

This function checks if the current argument should be skipped due to an exception or until the next separator or redirection.

Parameters

array	The original array of arguments.
new_array	The array to store formatted arguments.
i	The index of the current argument.
skip_until_separator	Pointer to the skip flag.

Returns

1 if skipping, 0 otherwise.

4.40.1.3 is_an_exception()

```
int is_an_exception (
            char ** array,
            char ** new_array,
            int i,
            int * skip_after_sep)
```

Checks for exceptions in an argument and ignores them.

If the argument is among which, where, alias or unalias, skip it

Parameters

array	The original array of arguments.
new_array	The array to store processed arguments.
i	The index of the current argument.
skip_after_sep	A pointer to the skip parameter.

Returns

1 if successful, -1 on error, or 0 if no exceptions are found.

4.40.1.4 is_separator_or_redirection()

```
int is_separator_or_redirection (
            const char * arg)
```

Checks if the argument is a separator or redirection.

This function determines if the given argument is one of the recognized separators or redirection operators.

Parameters

arg	The argument to check.

Returns

1 if the argument is a separator or redirection, 0 otherwise.

4.41 src/format/format.c File Reference

Functions to format and process shell input.

```
#include "globbings.h"
#include "meowsh.h"
#include "builtins.h"
#include "parser.h"
#include "exec.h"
#include "format.h"
```

Functions

- int check_quote (char **array, char **new_array, int i)
  Checks for quotes in an argument and processes them.
- char * format_input (char *input, shell_t *shell)
  Formats the input string for the shell.

4.41.1 Detailed Description

Functions to format and process shell input.

This file contains functions to handle exceptions in case of alias, unalias, which and where.

This file contains functions to handle aliases, environment variables, globbing, and history expansion for shell input. It processes user input to prepare it for execution in the shell.

4.41.2 Function Documentation

4.41.2.1 check_quote()

```
int check_quote (
            char ** array,
            char ** new_array,
            int i)
```

Checks for quotes in an argument and processes them.

If the argument is enclosed in single quotes, it removes the quotes and returns the processed string.

Parameters

array	The original array of arguments.
new_array	The array to store processed arguments.
i	The index of the current argument.

Returns

0 if successful, 1 on error, or -1 if no quotes are found.

4.41.2.2 format_input()

```
char * format_input (
            char * input,
            shell_t * shell)
```

Formats the input string for the shell.

Splits the input string into arguments, processes each argument for aliases, environment variables, globbing, and history expansion, and concatenates the results into a single formatted string.

Parameters

input	The input string to format.
shell	The shell structure containing aliases, environment variables, and history.

Returns

A newly allocated string containing the formatted input, or NULL on error.

4.42 src/format/put_spaces.c File Reference

Functions to add spaces around operators in a shell input string.

```
#include "format.h"
```

Functions

- char * add_spaces_around_operators (char *input)

  Adds spaces around operators in a shell input string.

4.42.1 Detailed Description

Functions to add spaces around operators in a shell input string.

This file contains functions to process a shell input string and ensure that spaces are added around operators (e.g., >, <, |, ;, >>, <<). It handles quoted strings to avoid modifying operators inside quotes.

4.42.2 Function Documentation

4.42.2.1 add_spaces_around_operators()

```
char * add_spaces_around_operators (
            char * input)
```

Adds spaces around operators in a shell input string.

This function processes the input string, identifies operators, and ensures that spaces are added around them. It handles quoted strings to avoid modifying operators inside quotes.

Parameters

input	The input string to process.

Returns

A newly allocated string with spaces added around operators, or NULL on memory allocation failure.

4.43 src/globbings/get_globbings.c File Reference

Functions for retrieving and managing globbing results.

```
#include "globbings.h"
```

Functions

- char * create_concat_array (char **pathv)

  Concatenate an array of strings into a single alloc'ed string.
- char * get_globbings (char *find)

  Retrieve globbing matches as a concatenated string.

4.43.1 Detailed Description

Functions for retrieving and managing globbing results.

This file implements helper functions to concatenate globbing search results based on a pattern.

4.43.2 Function Documentation

4.43.2.1 create_concat_array()

```
char * create_concat_array (
            char ** pathv)
```

Concatenate an array of strings into a single alloc'ed string.

This static function concatenates all strings from the provided array, separating each with a space. Memory is dynamically allocated for the new string and should be freed by the caller.

Parameters

pathv	Array of strings.

Returns

Pointer to the concatenated string, or NULL on failure.

4.43.2.2 get_globbings()

```
char * get_globbings (
            char * find)
```

Retrieve globbing matches as a concatenated string.

This function uses the glob() library function to find matches for the provided pattern. When matches are found, they are concatenated into a single string. The allocated memory must be freed by the caller.

Parameters

find	The globbing pattern.

Returns

Pointer to the concatenated result string, or NULL if no matches are found or on error.

4.44 src/globbings/is_gobbling.c File Reference

Function to check for globbing patterns in a string.

```
#include "globbings.h"
```

Functions

- int is_gobbling (char *str)
  Checks if a string contains globbing characters.

4.44.1 Detailed Description

Function to check for globbing patterns in a string.

This file contains the implementation of the `is_gobbling` function, which determines if a given string contains globbing characters such as `*`, `?`, `[`, or `]`.

4.44.2 Function Documentation

4.44.2.1 is_gobbling()

```
int is_gobbling (
            char * str)
```

Checks if a string contains globbing characters.

This function iterates through the input string and checks for the presence of globbing characters (`*`, `?`, `[`, `]`). If any of these characters are found, the function returns `1`. Otherwise, it returns `0`.

Parameters

str	The input string to check for globbing characters.

Returns

1 if the string contains globbing characters, 0 otherwise.

4.45 src/input/autocompletion.c File Reference

Implementation of the tab-autocompletion feature.

```
#include "input.h"
#include "globbings.h"
#include "parser.h"
```

Functions

- int autocomplete_tabs (input_t *input, shell_t *shell)
  Handles tab-autocompletion by invoking globbing logic.

4.45.1 Detailed Description

Implementation of the tab-autocompletion feature.

This file contains functions related to autocompletion based on globbing. It handles extracting the current word from user input, generating matches using glob patterns, and either completing the input with a unique match or displaying multiple possible matches.

4.45.2 Function Documentation

4.45.2.1 autocomplete_tabs()

```
int autocomplete_tabs (
            input_t * input,
            shell_t * shell)
```

Handles tab-autocompletion by invoking globbing logic.

This function processes the current input to provide tab-autocompletion. It extracts the current word being typed, constructs a globbing pattern, and uses it to find matches. If a single match is found, it replaces the word in the input. If multiple matches are found, it displays them on the terminal. The function also updates the input structure and redisplays the input line after processing.

Parameters

input	Pointer to the input structure containing the current input string, cursor position, and input length.
shell	Pointer to the shell structure containing environment variables and other shell-related data.

Returns

0 on success, 1 on memory allocation failure or other errors.

4.46 src/input/check_shortcut.c File Reference

Handles keyboard shortcuts for cursor movement and screen clearing.

```
#include "input.h"
```

Functions

- int check_shortcut (input_t *input)

  Handles keyboard shortcuts for cursor movement and screen clearing.

4.46.1 Detailed Description

Handles keyboard shortcuts for cursor movement and screen clearing.

This file contains the implementation of functions that process specific keyboard shortcuts such as `CTRL-A`, `CTRL-E`, `CTRL-B`, `CTRL-F`, `CTRL-T`, and `CTRL-L`. These shortcuts allow the user to navigate within the input line, transpose characters, or clear the screen.

4.46.2 Function Documentation

4.46.2.1 check_shortcut()

```
int check_shortcut (
            input_t * input)
```

Handles keyboard shortcuts for cursor movement and screen clearing.

This function checks for specific keyboard shortcuts:

- `CTRL-A` (ASCII 1): Moves the cursor to the beginning of the line.
- `CTRL-E` (ASCII 5): Moves the cursor to the end of the line.
- `CTRL-B` (ASCII 2): Moves the cursor backward by one character.
- `CTRL-F` (ASCII 6): Moves the cursor forward by one character.
- `CTRL-T` (ASCII 20): Swaps the last two characters in the input string.
- `CTRL-L` (ASCII 12): Clears the terminal screen and resets the cursor.

Parameters

input	Pointer to the input structure containing the current input string, its length, and the cursor position.

Returns

Always returns 0.

4.47 src/input/get_input.c File Reference

Implementation of the input handling for the Meowsh shell.

```
#include "input.h"
#include "builtins.h"
```

Functions

- int display_line (input_t *input, shell_t *shell)
  Displays the current input line in the terminal.
- char * get_input (shell_t *shell)
  Reads and processes user input in raw mode.

4.47.1 Detailed Description

Implementation of the input handling for the Meowsh shell.

This file contains functions to handle user input in raw mode, including cursor movement, character insertion, deletion, and line display.

4.47.2 Function Documentation

4.47.2.1 display_line()

```
int display_line (
            input_t * input,
            shell_t * shell)
```

Displays the current input line in the terminal.

This function clears the current line in the terminal, writes the input string, and repositions the cursor to the correct position.

Parameters

input	Pointer to the input structure containing the current input string, its length, and the cursor position.
shell	The struct containing all the data about the shell

Returns

0 on success.

4.47.2.2 get_input()

```
char * get_input (
            shell_t * shell)
```

Reads and processes user input in raw mode.

This function reads user input character by character, processes special keys, and updates the input string and cursor position accordingly.

Parameters

shell	Pointer to the shell structure (not used in this function but kept for consistency with the shell's architecture).

Returns

A dynamically allocated string containing the user input, or NULL on error.

4.48 src/input/handle_sequences.c File Reference

Implementation that handles user input sequences.

```
#include "input.h"
```

Functions

- int handle_sequences (input_t *input, shell_t *shell)

  Handles and redirects all sequences found in the user input.

4.48.1 Detailed Description

Implementation that handles user input sequences.

This file contains functions to handle user input sequences, such as cursor movement, history navigation, and special key handling.

4.48.2 Function Documentation

4.48.2.1 handle_sequences()

```
int handle_sequences (
            input_t * input,
            shell_t * shell)
```

Handles and redirects all sequences found in the user input.

This function detects if a sequence is found, reads the sequence, and redirects arrow key sequences to another function. It also handles the Suppr (Delete) key sequence directly.

Parameters

input	Pointer to the input structure containing the current input string, its length, and the cursor position.
shell	Pointer to the shell structure.

Returns

1 if a sequence is detected and handled, 0 otherwise.

4.49 src/input/manage_input.c File Reference

Implementation of the input management.

```
#include "input.h"
```

Functions

- int manage_input (input_t *input, shell_t *shell)
  Manages user input by handling special keys and printable characters.

4.49.1 Detailed Description

Implementation of the input management.

This file contains functions that manage user input character by character, including handling special keys (e.g., Backspace, Delete) and inserting printable characters into the input string.

4.49.2 Function Documentation

4.49.2.1 manage_input()

```
int manage_input (
            input_t * input,
            shell_t * shell)
```

Manages user input by handling special keys and printable characters.

This function processes user input character by character. It handles special keys (e.g., Backspace, Delete) and inserts printable characters into the input string. It also updates the input length and cursor position accordingly.

Parameters

input	Pointer to the input structure containing the current input string, its length, and the cursor position.
shell	Pointer to the shell structure containing environment variables and other shell-related data.

Returns

0 on success, 1 on memory allocation failure.

4.50 src/input/manage_input_struct.c File Reference

Functions to initialize and free the input structure.

```
#include "input.h"
```

Functions

- int init_input (input_t **input, shell_t *shell)

  Initializes the input structure.
- void free_input (input_t *input)

  Frees the memory allocated for the input structure.

4.50.1 Detailed Description

Functions to initialize and free the input structure.

This file contains functions to manage the lifecycle of the `input_t` structure, including initialization and memory cleanup.

4.50.2 Function Documentation

4.50.2.1 free_input()

```
void free_input (
            input_t * input)
```

Frees the memory allocated for the input structure.

This function frees the memory allocated for the `input_t` structure, including the dynamically allocated input string.

Parameters

input	Pointer to the input structure to free.

4.50.2.2 init_input()

```
int init_input (
            input_t ** input,
            shell_t * shell)
```

Initializes the input structure.

This function allocates memory for the `input_t` structure and initializes its fields to default values.

Parameters

input	Double pointer to the input structure to initialize.
shell	Pointer to the shell structure containing environment variables and other shell-related data.

Returns

0 on success, 1 on memory allocation failure.

4.51 src/parsing/command_handling/open_fd_redirections.c File Reference

Creates the fd for input and output based off redirections.

```
#include "parser.h"
```

Functions

- int open_input_fd (commands_t *cmd, char *filename)
  Opens the input file descriptor for a command.
- int open_output_fd (commands_t *cmd, bool truncate, char *filename)
  Opens the output file descriptor for a command.

4.51.1 Detailed Description

Creates the fd for input and output based off redirections.

4.51.2 Function Documentation

4.51.2.1 open_input_fd()

```
int open_input_fd (
            commands_t * cmd,
            char * filename)
```

Opens the input file descriptor for a command.

Parameters

cmd	Pointer to the command structure (commands_t).
filename	The path to the file to open as input.

Returns

The new input file descriptor, or -1 on failure.

4.51.2.2 open_output_fd()

```
int open_output_fd (
            commands_t * cmd,
            bool truncate,
            char * filename)
```

Opens the output file descriptor for a command.

Parameters

cmd	Pointer to the command structure (commands_t).
truncate	If true, the file is truncated; if false, data is appended.
filename	The path to the file to open as output.

Returns

The new output file descriptor, or -1 on failure.

4.52 src/parsing/command_handling/separators.c File Reference

Gets the highest-level separator between ';', '&&', '||' and '|'.

```
#include "parser.h"
```

Functions

- sep_t change_separator (sep_t highest_separator, char *line, int i)
  Update the highest separator found in the line.
- sep_t get_highest_separator (char *line)
  Get the highest-priority separator in the line.
- int is_simple_separator (char current_character, sep_t separator)
  Checks if the current character is a separator.
- int is_double_separator (char *line, int *current, sep_t separator, bool does_increment)
  Checks if the current and following character are separators.

4.52.1 Detailed Description

Gets the highest-level separator between ';', '&&', '||' and '|'.

4.52.2 Function Documentation

4.52.2.1 change_separator()

```
sep_t change_separator (
            sep_t highest_separator,
            char * line,
            int i)
```

Update the highest separator found in the line.

Compares the current highest and the one found at position i.

Parameters

highest_separator	The current highest separator.
line	The input command line.
i	Current index being checked.

Returns

The updated highest separator.

4.52.2.2 get_highest_separator()

```
sep_t get_highest_separator (
            char * line)
```

Get the highest-priority separator in the line.

Skips nested structures like parentheses, quotes and backticks.

Parameters

line	The input command line.

Returns

The most important separator found, or -1 if none.

4.52.2.3 is_double_separator()

```
int is_double_separator (
            char * line,
            int * current,
            sep_t separator,
            bool does_increment)
```

Checks if the current and following character are separators.

Only in case of AND/OR separator.

Parameters

line	The input command line.
current	The current character index.
separator	The highest sep_t defined separator.
does_increment	true if current has to be increased, else false.

Returns

1 if it's a separator, else 0.

4.52.2.4 is_simple_separator()

```
int is_simple_separator (
            char current_character,
            sep_t separator)
```

Checks if the current character is a separator.

Only in case of semicolumn and pipe.

Parameters

current_character	The current character to compare.
separator	The highest sep_t defined separator.

Returns

1 if it's a separator, else 0.

4.53 src/parsing/command_handling/simple_command.c File Reference

Handles an individual command, with input and output redirections.

```
#include "parser.h"
#include "utilities.h"
```

Functions

- commands_t * create_command_node (void)
  Initialize a new command node.
- commands_t * add_command (char *line, commands_t *commands)
  Add a command to the current commands_t node.
- commands_t * check_input_redirection (char *line, commands_t *commands, int i)
  Checks for input redirection in the command line and processes it.
- commands_t * check_output_redirection (char *line, commands_t *commands, int *i)
  Checks for output redirection in the command line and processes it.
- commands_t * handle_no_separator_command (char *line, commands_t *commands)
  Handle parsing when no separator is found.

4.53.1 Detailed Description

Handles an individual command, with input and output redirections.

4.53.2 Function Documentation

4.53.2.1 add_command()

```
commands_t * add_command (
            char * line,
            commands_t * commands)
```

Add a command to the current commands_t node.

Creates a subcommand and prepares it for parsing.

Parameters

line	The command line to parse.
commands	The parent command node to attach to.

Returns

Pointer to the subcommand created, or NULL on error.

4.53.2.2 check_input_redirection()

```
commands_t * check_input_redirection (
             char * line,
             commands_t * commands,
             int i)
```

Checks for input redirection in the command line and processes it.

This function parses a possible input redirection ('<') in the line, handles heredoc redirection ('<<'), and opens the corresponding input file for standard input redirection.

Parameters

line	The command line string being parsed (will be modified).
commands	The structure to store the input file descriptor or heredoc.
i	The current index in the line where the '<' character was found.

Returns

A pointer to the updated commands structure, or NULL on error.

4.53.2.3 check_output_redirection()

```
commands_t * check_output_redirection (
             char * line,
             commands_t * commands,
             int * i)
```

Checks for output redirection in the command line and processes it.

This function handles both simple (>) and append (>>) output redirection. It opens the corresponding output file and stores its file descriptor in `commands->fd_output`.

Parameters

line	The command line string being parsed (will be modified).
commands	The command structure to store the output fd in
i	A pointer to the current index in the line where the '>' was found

Returns

A pointer to the updated commands structure, or NULL on error.

4.53.2.4 create_command_node()

```
commands_t * create_command_node (
             void )
```

Initialize a new command node.

Allocates and initializes a new commands_t structure with default values.

Returns

Pointer to the newly created commands_t, or NULL on failure.

4.53.2.5 handle_no_separator_command()

```
commands_t * handle_no_separator_command (
             char * line,
             commands_t * commands)
```

Handle parsing when no separator is found.

If the command is nested in parentheses, remove them and parse again. Otherwise, just add the command directly.

Parameters

line	The input command line.
commands	The current commands_t node.

Returns

The updated commands_t after handling the command.

4.54 src/parsing/is_command_valid/check_double_and.c File Reference

Functions to check the validity of &&' syntax in the line.

```
#include "parser.h"
```

Functions

- int is_valid_and_syntax (char *line)

  Checks if the && syntax in the command line is valid.

4.54.1 Detailed Description

Functions to check the validity of &&' syntax in the line.

4.54.2 Function Documentation

4.54.2.1 is_valid_and_syntax()

```
int is_valid_and_syntax (
            char * line)
```

Checks if the && syntax in the command line is valid.

This function ensures that:

- The line does not start with &&.
- && are not followed by invalid or missing commands.
- && checks ignore characters inside quotes.

Parameters

line	The command line string to validate.

Returns

1 if the && syntax is valid, 0 otherwise.

4.55 src/parsing/is_command_valid/check_matched.c File Reference

Check if parenthesis and quotes are matched.

```
#include "parser.h"
```

Functions

- bool is_nesting_matched (char *line)
  Checks if quotes and parentheses are matched in the input line.

4.55.1 Detailed Description

Check if parenthesis and quotes are matched.

4.55.2 Function Documentation

4.55.2.1 is_nesting_matched()

```
bool is_nesting_matched (
            char * line)
```

Checks if quotes and parentheses are matched in the input line.

Parameters

line	The input string to validate.

Returns

true if all quotes and parentheses are closed, false otherwise.

4.56 src/parsing/is_command_valid/check_pipes.c File Reference

Functions to check the validity of pipes' syntax in the line.

```
#include "parser.h"
```

Functions

- bool is_space (char c)

  Checks if a character is a space, tab, or newline.
- void update_quote_state (char c, bool *in_single_quote, bool *in_double_quote)

  Updates the quote states depending on the current character.
- bool is_null_command (const char *line, int i)

  Determines if a null command follows the current pipe character.
- int is_valid_pipe_syntax (char *line)

  Checks if the pipe syntax in the command line is valid.

4.56.1 Detailed Description

Functions to check the validity of pipes' syntax in the line.

4.56.2 Function Documentation

4.56.2.1 is_null_command()

```
bool is_null_command (
            const char * line,
            int i)
```

Determines if a null command follows the current pipe character.

Skips spaces and checks if there's nothing valid after the pipe.

Parameters

line	The command line string.
i	The index of the pipe character.

Returns

true if nothing follows the pipe, false otherwise.

4.56.2.2 is_space()

```
bool is_space (
            char c)
```

Checks if a character is a space, tab, or newline.

Parameters

c	The character to check.

Returns

true if the character is a whitespace, false otherwise.

4.56.2.3 is_valid_pipe_syntax()

```
int is_valid_pipe_syntax (
            char * line)
```

Checks if the pipe syntax in the command line is valid.

This function ensures that:

- The line does not start with a pipe.
- Pipes are not followed by invalid or missing commands.
- Pipe checks ignore characters inside quotes.

Parameters

line	The command line string to validate.

Returns

1 if the pipe syntax is valid, 0 otherwise.

4.56.2.4 update_quote_state()

```
void update_quote_state (
            char c,
            bool * in_single_quote,
            bool * in_double_quote)
```

Updates the quote states depending on the current character.

This function toggles the in_single_quote or in_double_quote flags depending on whether the current character is a quote and whether we are currently inside the other type of quote.

Parameters

c	The current character.
in_single_quote	Pointer to a flag : if inside single quotes.
in_double_quote	Pointer to a flag : if inside double quotes.

4.57 src/parsing/is_command_valid/check_semicolons.c File Reference

Functions to check the validity of ';" syntax in the line.

```
#include "parser.h"
```

Functions

- int is_valid_semicolon_syntax (char *line)
  Checks if the ; syntax in the command line is valid.

4.57.1 Detailed Description

Functions to check the validity of ';" syntax in the line.

4.57.2 Function Documentation

4.57.2.1 is_valid_semicolon_syntax()

```
int is_valid_semicolon_syntax (
            char * line)
```

Checks if the ; syntax in the command line is valid.

This function ensures that:

- The line does not start with ;.
- ; are not followed by invalid or missing commands.
- ; checks ignore characters inside quotes.

Parameters

line	The command line string to validate.

Returns

1 if the ; syntax is valid, 0 otherwise.

4.58 src/parsing/is_command_valid/is_command_valid.c File Reference

Ensures that the command line is properly formatted before execution.

```
#include "parser.h"
```

Functions

- int is_valid_syntax (char *line)
  Checks the line for errors.

4.58.1 Detailed Description

Ensures that the command line is properly formatted before execution.

Handles nesting, and changes it if encountering parenthesis or quotes.

4.58.2 Function Documentation

4.58.2.1 is_valid_syntax()

```
int is_valid_syntax (
            char * line)
```

Checks the line for errors.

Validates the syntax.

Parameters

line	The command line to check.

Returns

1 if the line is well formated, else 0.

4.59 src/parsing/main_parsing/nesting_handling.c File Reference

```
#include "parser.h"
```

Functions

- void change_depth (parsing_state_t *parsing_elements, char current_character)
  Modify the depth counter when encountering parentheses.
- int is_nested (parsing_state_t parsing_elements)
  Check if the parser is currently in a nested state.
- void change_backtick_and_quotes (parsing_state_t *parsing_elements, char current_character)
  Toggle quote or backtick status depending on the character.
- int is_command_inside_parenthesis (char *line)
  Check if the command is entirely wrapped in parentheses.
- char * cut_parenthesis (char *line)
  Remove outer parentheses from the command line.

4.59.1 Function Documentation

4.59.1.1 change_backtick_and_quotes()

```
void change_backtick_and_quotes (
            parsing_state_t * parsing_elements,
            char current_character)
```

Toggle quote or backtick status depending on the character.

Parameters

parsing_elements	Pointer to the parsing elements' state
current_character	Current character being read.

4.59.1.2 change_depth()

```
void change_depth (
            parsing_state_t * parsing_elements,
            char current_character)
```

Modify the depth counter when encountering parentheses.

Increments or decrements the depth counter depending on the character.

Parameters

parsing_elements	Pointer to the parsing elements' state
current_character	Current character being read.

4.59.1.3 cut_parenthesis()

```
char * cut_parenthesis (
            char * line)
```

Remove outer parentheses from the command line.

Allocates and returns a copy of the line without outer parentheses.

Parameters

line	The input command line.

Returns

A new string without the outer parentheses, or NULL on failure.

4.59.1.4 is_command_inside_parenthesis()

```
int is_command_inside_parenthesis (
            char * line)
```

Check if the command is entirely wrapped in parentheses.

Parameters

line	The input command line.

Returns

1 if command is parenthesized, 0 otherwise.

4.59.1.5 is_nested()

```
int is_nested (
            parsing_state_t parsing_elements)
```

Check if the parser is currently in a nested state.

Determines if the parser is inside quotes, backticks or parentheses.

Parameters

parsing_elements	The parsing elements' state

Returns

1 if nested, 0 otherwise.

4.60 src/parsing/main_parsing/parser.c File Reference

Main parsing functions that recursively parse into subcommands.

```
#include "parser.h"
```

Functions

- int get_number_subcommands (sep_t separator, char *line)
  Get the number of subcommands in a command.
- char * cut_substring (char *line, int *start_index, int index, int is_simple_separator)
  Parse a full command line recursively.
- commands_t * parse_commands (char *line, commands_t *commands, sep_t separator)
  Parse a full command line recursively.
- commands_t * choose_parsing (char *line, commands_t *commands, sep_t separator, int depth)
  Parses a command line and constructs a command structure based on separators.
- commands_t * parse_line (char *line)
  Entry point for parsing a command line.

4.60.1 Detailed Description

Main parsing functions that recursively parse into subcommands.

4.60.2 Function Documentation

4.60.2.1 choose_parsing()

```
commands_t * choose_parsing (
            char * line,
            commands_t * commands,
            sep_t separator,
            int depth)
```

Parses a command line and constructs a command structure based on separators.

This function analyzes the input line, handles command separators (e.g., `;`, `&&`, `||`), and splits the command into subcommands. It also manages the depth for nested commands (such as parentheses) and assigns the good separator to each subcommand.

Parameters

line	The input command line.
commands	The parent command node.
separator	The separator to use if needed to cut
depth	The current depth, used for parenthesis handling

Returns

A pointer to the new added command, or NULL on error.

4.60.2.2 cut_substring()

```
char * cut_substring (
            char * line,
            int * start_index,
            int index,
            int is_simple_separator)
```

Parse a full command line recursively.

Splits the line by the highest-level separator and recursively parses subcommands.

Parameters

line	The input command line.
start_index	A pointer to the start index of the line.
index	The end index of the line.
is_simple_separator	1 if the separator is COLUMN or PIPE, else 0.

Returns

A pointer to the new substring, NULL on error.

4.60.2.3 get_number_subcommands()

```
int get_number_subcommands (
            sep_t separator,
            char * line)
```

Get the number of subcommands in a command.

Counts the number of subcommands using a separator.

Parameters

separator	The separator to cut my line.
line	The input command line.

Returns

The number of subcommands.

4.60.2.4 parse_commands()

```
commands_t * parse_commands (
             char * line,
             commands_t * commands,
             sep_t separator)
```

Parse a full command line recursively.

Splits the line by the highest-level separator and recursively parses subcommands.

Parameters

line	The input command line.
commands	The parent command node.
separator	The separator to use to cut the line

Returns

A pointer to the new added command, or NULL on error.

4.60.2.5 parse_line()

```
commands_t * parse_line (
             char * line)
```

Entry point for parsing a command line.

Validates the syntax and initializes the command tree.

Parameters

line	The command line to parse.

Returns

The root of the parsed command tree, or NULL on error.

4.61 src/utilities/free_shell.c File Reference

Functions to free memory allocated for the shell structure.

```
#include "utilities.h"
```

Functions

- void free_commands (commands_t **cmds)

  Frees memory allocated for an array of commands.
- void free_aliases (shell_t *shell)

  Frees memory allocated for the shell's aliases.
- void free_shell (shell_t *shell)

  Frees all memory allocated for the shell structure.

4.61.1 Detailed Description

Functions to free memory allocated for the shell structure.

This file contains functions to free all dynamically allocated memory associated with the `shell_t` structure, including commands, environment variables, aliases, and history.

4.61.2 Function Documentation

4.61.2.1 free_aliases()

```
void free_aliases (
            shell_t * shell)
```

Frees memory allocated for the shell's aliases.

This function iterates through the linked list of aliases and frees each alias structure, including the original and new name strings.

Parameters

shell	Pointer to the shell structure containing the aliases.

4.61.2.2 free_commands()

```
void free_commands (
            commands_t ** cmds)
```

Frees memory allocated for an array of commands.

This function iterates through an array of `commands_t` structures and frees all associated memory, including file descriptors, heredoc strings, paths, commands, and subcommands.

Parameters

cmds	Double pointer to the array of commands to free.

4.61.2.3 free_shell()

```
void free_shell (
            shell_t * shell)
```

Frees all memory allocated for the shell structure.

This function frees all dynamically allocated memory associated with the `shell_t` structure, including commands, environment variables, aliases, history, and the old working directory.

Parameters

shell	Pointer to the shell structure to free.

4.62 src/utilities/init_shell.c File Reference

Initialization functions for the shell program.

```
#include "utilities.h"
#include "builtins.h"
```

Functions

- shell_t * init_struct (shell_t *shell, char **envp)

  Initializes a shell structure.

4.62.1 Detailed Description

Initialization functions for the shell program.

This file contains the implementation of functions responsible for initializing the shell structure and its components, including commands, environment variables, aliases, and command history.

The primary function in this file is `init_struct`, which allocates memory for the shell structure and initializes its fields based on the provided environment variables.

4.62.2 Function Documentation

4.62.2.1 init_struct()

```
shell_t * init_struct (
            shell_t * shell,
            char ** envp)
```

Initializes a shell structure.

This function allocates memory for a shell_t structure and its associated components, including commands, environment variables, aliases, and history. It also initializes the shell's working directory and exit status.

Parameters

shell	A pointer to a shell_t structure. If NULL, a new structure will be allocated and initialized.
envp	An array of environment variable strings. This will be used to populate the global environment variables in the shell.

Returns

A pointer to the initialized shell_t structure on success, or NULL if memory allocation fails or if any of the components cannot be initialized.

Note

The caller is responsible for freeing the allocated memory for the shell_t structure and its components when they are no longer needed.

4.63 src/utilities/int_to_str.c File Reference

```
#include "utilities.h"
```

Functions

- char * int_to_str (int value)

  Converts a non-negative integer to a dynamically allocated string.

4.63.1 Function Documentation

4.63.1.1 int_to_str()

```
char * int_to_str (
            int value)
```

Converts a non-negative integer to a dynamically allocated string.

Allocates memory to store the string representation of the given integer and formats it into the allocated memory.

Parameters

value	The integer value to convert (must be non-negative).

Returns

A dynamically allocated string containing the number representation, or NULL on allocation failure or invalid input.

4.64 src/utilities/manage_char_array.c File Reference

Utility functions to manage arrays of strings.

```
#include "utilities.h"
```

Functions

- void free_array (char **array)

  Frees a dynamically allocated array of strings.
- char ** dup_array (char **src)

  Duplicates an array of strings.
- int del_elt (char ***array, int index)

  Deletes an element from an array of strings.

4.64.1 Detailed Description

Utility functions to manage arrays of strings.

This file contains helper functions to manage dynamically allocated arrays of strings, including freeing, duplicating, and deleting elements.

4.64.2 Function Documentation

4.64.2.1 del_elt()

```
int del_elt (
            char *** array,
            int index)
```

Deletes an element from an array of strings.

This function removes the element at the specified index from the array and shifts the remaining elements to fill the gap. The array is resized accordingly.

Parameters

array	Pointer to the array of strings.
index	The index of the element to delete.

Returns

0 on success, 1 on failure (e.g., invalid index or memory allocation failure).

4.64.2.2 dup_array()

```
char ** dup_array (
            char ** src)
```

Duplicates an array of strings.

This function creates a new array of strings that is a deep copy of the source array. Each string in the source array is duplicated.

Parameters

src	The source array of strings to duplicate.

Returns

A newly allocated array of strings, or NULL on failure.

4.64.2.3 free_array()

```
void free_array (
            char ** array)
```

Frees a dynamically allocated array of strings.

This function iterates through the array, freeing each string, and then frees the array itself.

Parameters

array	The array of strings to free.

4.65 src/utilities/my_str_to_word_array.c File Reference

Functions to split a string into an array of words.

```
#include "parser.h"
```

Functions

- char ** my_str_to_word_array (const char *str, const char *spliters)

  Splits a string into an array of words.

4.65.1 Detailed Description

Functions to split a string into an array of words.

This file contains the implementation of the `my_str_to_word_array` function, which splits a string into an array of words based on specified delimiters. It also handles quoted strings to ensure they are treated as a single word.

4.65.2 Function Documentation

4.65.2.1 my_str_to_word_array()

```
char ** my_str_to_word_array (
            const char * str,
            const char * spliters)
```

Splits a string into an array of words.

This function splits the input string into an array of words based on the specified delimiters. Quoted sections are treated as a single word.

Parameters

str	The input string to split.
spliters	A string containing the delimiter characters.

Returns

A dynamically allocated array of strings, or NULL on failure.

4.66 src/utilities/str_alphanum.c File Reference

Utility function to check if a string is alphanumeric.

```
#include "builtins.h"
```

Functions

- int str_isalphanum (char *str)
  Checks if a string is alphanumeric.

4.66.1 Detailed Description

Utility function to check if a string is alphanumeric.

This file contains the implementation of the `str_isalphanum` function, which verifies if a given string contains only alphanumeric characters.

4.66.2 Function Documentation

4.66.2.1 str_isalphanum()

```
int str_isalphanum (
            char * str)
```

Checks if a string is alphanumeric.

This function iterates through the input string and checks if all characters are alphanumeric (letters or digits). If any character is not alphanumeric, the function returns `0`. Otherwise, it returns `1`.

Parameters

str	The input string to check.

Returns

1 if the string is alphanumeric, 0 otherwise.

4.67 src/exec/strjoin.c File Reference

Function to concatenate two strings.

```
#include "utilities.h"
```

Functions

- char * strjoin (char *str1, char *str2)

  Concatenate two strings into a new string.

4.67.1 Detailed Description

Function to concatenate two strings.

This file contains the implementation of the `strjoin` function, which concatenates two strings into a newly allocated string.

4.67.2 Function Documentation

4.67.2.1 strjoin()

```
char * strjoin (
            char * str1,
            char * str2)
```

Concatenate two strings into a new string.

This function takes two input strings, allocates memory for a new string that contains the concatenation of the two, and returns the new string.

Parameters

str1	The first string to concatenate.
str2	The second string to concatenate.

Returns

A dynamically allocated string containing the concatenation of `str1` and `str2`, or `NULL` if memory allocation fails.

4.68 src/utilities/strjoin.c File Reference

Function to concatenate two strings.

```
#include "utilities.h"
```

Functions

- char * strjoin (char *str1, char *str2)

  Concatenate two strings into a new string.

4.68.1 Detailed Description

Function to concatenate two strings.

This file contains the implementation of the `strjoin` function, which concatenates two strings into a newly allocated string.

4.68.2 Function Documentation

4.68.2.1 strjoin()

```
char * strjoin (
            char * str1,
            char * str2)
```

Concatenate two strings into a new string.

This function takes two input strings, allocates memory for a new string that contains the concatenation of the two, and returns the new string.

Parameters

str1	The first string to concatenate.
str2	The second string to concatenate.

Returns

A dynamically allocated string containing the concatenation of `str1` and `str2`, or `NULL` if memory allocation fails.

Index