42sh

Generated by Doxygen 1.13.2

1	Class Index	1
	1.1 Class List	1
2	File Index	3
	2.1 File List	3
3	Class Documentation	7
	3.1 aliases_s Struct Reference	7
	3.1.1 Member Data Documentation	7
	3.1.1.1 browsed	7
	3.1.1.2 new_name	7
	3.1.1.3 next	7
	3.1.1.4 original	7
	3.2 commands_s Struct Reference	8
	3.2.1 Member Data Documentation	8
	3.2.1.1 command	8
	3.2.1.2 depth	8
	3.2.1.3 fd_input	8
	3.2.1.4 fd_output	8
	3.2.1.5 heredoc	8
	3.2.1.6 path	8
	3.2.1.7 separator	8
	3.2.1.8 subcommands	9
	3.3 env_s Struct Reference	9
	3.3.1 Member Data Documentation	9
	3.3.1.1 global	9
	3.3.1.2 local	9
	3.4 errno_outputs Struct Reference	9
	3.4.1 Member Data Documentation	10
	3.4.1.1 errmsg	10
	3.4.1.2 errnum	10
	3.5 exec_data_s Struct Reference	10
	3.5.1 Member Data Documentation	10
	3.5.1.1 cmd_type	10
	3.5.1.2 nb_elt	10
	3.5.1.3 pid	10
	3.5.1.4 pipe	10
	3.5.1.5 prev_depth	11
	3.5.1.6 status	11
	3.6 input_s Struct Reference	11
	3.6.1 Member Data Documentation	11
	3.6.1.1 char_input	11
	3.6.1.2 cursor_pos	11

3.6.1.3 history_pos	 11
3.6.1.4 input_len	 11
3.6.1.5 stock_history	 12
3.6.1.6 str_input	 12
3.7 parsing_state_s Struct Reference	 12
3.7.1 Member Data Documentation	 12
3.7.1.1 current_depth	 12
3.7.1.2 current_subcommand_index	 12
3.7.1.3 in_backtick	 12
3.7.1.4 in_double_quote	 12
3.7.1.5 in_single_quote	 13
3.7.1.6 line	 13
3.7.1.7 separator	 13
3.7.1.8 start_index	 13
3.8 shell_s Struct Reference	 13
3.8.1 Member Data Documentation	 13
3.8.1.1 aliases	 13
3.8.1.2 commands	 13
3.8.1.3 env	
3.8.1.4 exit_status	 14
3.8.1.5 history	 14
3.8.1.6 history_size	 14
3.8.1.7 owd	
3.8.1.8 run	
3.9 tab_builtins_t Struct Reference	 14
3.9.1 Member Data Documentation	 14
3.9.1.1 builtin_func	 14
3.9.1.2 command	 15
3.9.1.3 local	 15
4 File Documentation	17
4.1 include/builtins.h File Reference	
4.1.1 Macro Definition Documentation	
4.1.1.1 BUFFER_SIZE	
4.1.1.2 NB_BUILT_IN	
4.1.2 Function Documentation	
4.1.2.1 add_elt()	
4.1.2.2 add_node()	
4.1.2.3 add_to_history()	
4.1.2.4 delete_node()	
4.1.2.5 display_history()	
4.1.2.6 exec_history()	

4.1.2.7 free_history()	 20
4.1.2.8 free_node()	 21
4.1.2.9 get_history()	 21
4.1.2.10 get_root_home()	 21
4.1.2.11 get_value_key()	 22
4.1.2.12 handle_tilde()	 22
4.1.2.13 interpret_alias()	 22
4.1.2.14 interpret_unalias()	 23
4.1.2.15 interpret_where()	 23
4.1.2.16 interpret_which()	 24
4.1.2.17 is_a_builtin_command_where()	 24
4.1.2.18 is_a_builtin_command_which()	 24
4.1.2.19 my_cd()	 25
4.1.2.20 my_env()	 25
4.1.2.21 my_exit()	 26
4.1.2.22 my_setenv()	 26
4.1.2.23 my_unsetenv()	 26
4.2 builtins.h	 27
4.3 include/error_outputs.h File Reference	 28
4.3.1 Typedef Documentation	 28
4.3.1.1 errno_outputs_t	 28
4.3.2 Variable Documentation	 28
4.3.2.1 errno_table	 28
4.4 error_outputs.h	 29
4.5 include/exec.h File Reference	 29
4.5.1 Macro Definition Documentation	 30
4.5.1.1 BUILT_CMD	 30
4.5.1.2 CMD	 30
4.5.1.3 NOTHING	 30
4.5.1.4 SUBCMD	 30
4.5.2 Typedef Documentation	 31
4.5.2.1 exec_data_t	 31
4.5.3 Function Documentation	 31
4.5.3.1 check_path()	 31
4.5.3.2 check_signal()	 31
4.5.3.3 exec_cmd()	 31
4.5.3.4 free_exec_data()	 32
4.5.3.5 get_cmd_path()	 32
4.5.3.6 get_cmd_type()	 32
4.5.3.7 get_paths()	 33
4.5.3.8 init_exec_data()	 33
4.5.3.9 manage_saved_fd()	 34

4.6 exec.h	35
4.7 include/format.h File Reference	35
4.7.1 Macro Definition Documentation	36
4.7.1.1 MAX_NBR	36
4.7.1.2 NB_EXCEPTIONS	36
4.7.2 Function Documentation	36
4.7.2.1 add_spaces_around_operators()	36
4.7.2.2 check_quote()	37
4.7.2.3 format_input()	37
4.7.2.4 handle_exceptions_and_quotes()	38
4.7.2.5 handle_skipping()	39
4.8 format.h	39
4.9 include/globbings.h File Reference	40
4.9.1 Macro Definition Documentation	40
4.9.1.1 MAX_GLOBS	40
4.9.2 Function Documentation	40
4.9.2.1 create_concat_array()	40
4.9.2.2 get_globbings()	41
4.9.2.3 is_gobbling()	41
4.10 globbings.h	42
4.11 include/input.h File Reference	42
4.11.1 Macro Definition Documentation	43
4.11.1.1 BLUE	43
4.11.1.2 PURPLE	43
4.11.1.3 RESET	43
4.11.2 Typedef Documentation	43
4.11.2.1 input_t	43
4.11.3 Function Documentation	43
4.11.3.1 autocomplete_tabs()	43
4.11.3.2 check_shortcut()	44
4.11.3.3 display_line()	44
4.11.3.4 free_input()	45
4.11.3.5 get_input()	45
4.11.3.6 handle_sequences()	45
4.11.3.7 init_input()	46
4.11.3.8 manage_input()	46
4.12 input.h	47
4.13 include/meowsh.h File Reference	47
4.13.1 Typedef Documentation	48
4.13.1.1 aliases_t	48
4.13.1.2 commands_t	48
4.13.1.3 env_t	48

4.13.1.4 sep_t	48
4.13.1.5 shell_t	48
4.13.2 Enumeration Type Documentation	48
4.13.2.1 sep_s	48
4.13.3 Function Documentation	49
4.13.3.1 str_isalphanum()	49
4.14 meowsh.h	49
4.15 include/parser.h File Reference	50
4.15.1 Typedef Documentation	51
4.15.1.1 parsing_state_t	51
4.15.2 Function Documentation	52
4.15.2.1 add_command()	52
4.15.2.2 change_backtick_and_quotes()	52
4.15.2.3 change_depth()	52
4.15.2.4 change_separator()	53
4.15.2.5 check_input_redirection()	54
4.15.2.6 check_output_redirection()	54
4.15.2.7 choose_parsing()	55
4.15.2.8 create_command_node()	
4.15.2.9 cut_parenthesis()	55
4.15.2.10 get_highest_separator()	56
4.15.2.11 get_number_subcommands()	56
4.15.2.12 handle_no_separator_command()	56
4.15.2.13 is_command_inside_parenthesis()	57
4.15.2.14 is_double_separator()	58
4.15.2.15 is_nested()	58
4.15.2.16 is_nesting_matched()	58
4.15.2.17 is_null_command()	59
4.15.2.18 is_simple_separator()	59
4.15.2.19 is_space()	59
4.15.2.20 is_valid_and_syntax()	60
4.15.2.21 is_valid_pipe_syntax()	60
4.15.2.22 is_valid_semicolon_syntax()	61
4.15.2.23 is_valid_syntax()	61
4.15.2.24 open_input_fd()	61
4.15.2.25 open_output_fd()	62
4.15.2.26 parse_commands()	62
4.15.2.27 parse_line()	62
4.15.2.28 update_quote_state()	63
4.16 parser.h	63
4.17 include/utilities.h File Reference	64
4.17.1 Function Documentation	65

4.17.1.1 del_elt()	65
4.17.1.2 dup_array()	65
4.17.1.3 free_array()	65
4.17.1.4 free_commands()	66
4.17.1.5 free_shell()	66
4.17.1.6 init_struct()	66
4.17.1.7 int_to_str()	67
4.17.1.8 my_str_to_word_array()	67
4.17.1.9 strjoin()	67
4.18 utilities.h	68
4.19 src/builtins/alias/alias.c File Reference	68
4.19.1 Detailed Description	68
4.19.2 Function Documentation	69
4.19.2.1 interpret_alias()	69
4.20 src/builtins/alias/handle_alias.c File Reference	69
4.20.1 Detailed Description	69
4.20.2 Function Documentation	70
4.20.2.1 add_node()	70
4.20.2.2 delete_node()	70
4.20.2.3 free_node()	70
4.21 src/builtins/alias/unalias.c File Reference	71
4.21.1 Detailed Description	71
4.21.2 Function Documentation	71
4.21.2.1 interpret_unalias()	71
4.22 src/builtins/cd/cd_tilde.c File Reference	71
4.22.1 Detailed Description	72
4.22.2 Function Documentation	72
4.22.2.1 handle_tilde()	72
4.23 src/builtins/cd/get_home_user.c File Reference	72
4.23.1 Detailed Description	73
4.23.2 Function Documentation	73
4.23.2.1 get_root_home()	73
4.24 src/builtins/cd/my_cd.c File Reference	73
4.24.1 Detailed Description	73
4.24.2 Function Documentation	73
4.24.2.1 my_cd()	73
4.25 src/builtins/cd/my_getenv.c File Reference	74
4.25.1 Detailed Description	74
4.25.2 Function Documentation	74
4.25.2.1 get_value_key()	74
4.26 src/builtins/env/env.c File Reference	75
4 26 1 Detailed Description	75

4.26.2 Function Documentation	75
4.26.2.1 my_env()	75
4.27 src/builtins/env/setenv.c File Reference	75
4.27.1 Detailed Description	76
4.27.2 Function Documentation	76
4.27.2.1 add_env()	76
4.27.2.2 change_env()	76
4.27.2.3 get_line_env()	77
4.27.2.4 is_a_good_var()	77
4.27.2.5 my_setenv()	77
4.28 src/builtins/env/unsetenv.c File Reference	78
4.28.1 Detailed Description	78
4.28.2 Function Documentation	78
4.28.2.1 add_elt()	78
4.28.2.2 my_unsetenv()	79
4.29 src/builtins/exec/my_exit.c File Reference	79
4.29.1 Detailed Description	79
4.29.2 Function Documentation	79
4.29.2.1 my_exit()	79
4.30 src/builtins/history/handle_history.c File Reference	80
4.30.1 Detailed Description	80
4.30.2 Function Documentation	80
4.30.2.1 add_to_history()	80
4.30.2.2 display_history()	81
4.30.2.3 exec_history()	81
4.30.2.4 free_history()	81
4.30.2.5 get_history()	82
4.31 src/builtins/which/which.c File Reference	82
4.31.1 Detailed Description	82
4.31.2 Function Documentation	82
4.31.2.1 interpret_which()	82
4.32 src/builtins/which_and_where/which.c File Reference	83
4.32.1 Detailed Description	83
4.32.2 Function Documentation	83
4.32.2.1 interpret_which()	83
4.32.2.2 is_a_builtin_command_which()	84
4.33 src/builtins/which_and_where/where.c File Reference	84
4.33.1 Detailed Description	84
4.33.2 Function Documentation	85
4.33.2.1 interpret_where()	85
4.33.2.2 is_a_builtin_command_where()	85
4.34 src/exec/check_signal.c File Reference	85

4.34.1 Detailed Description	. 86
4.34.2 Function Documentation	. 86
4.34.2.1 check_signal()	. 86
4.35 src/exec/exec_cmd.c File Reference	. 86
4.35.1 Detailed Description	. 87
4.35.2 Function Documentation	. 87
4.35.2.1 exec_cmd()	. 87
4.36 src/exec/get_cmd_path.c File Reference	. 87
4.36.1 Detailed Description	. 87
4.36.2 Function Documentation	. 88
4.36.2.1 check_path()	. 88
4.36.2.2 get_cmd_path()	. 88
4.36.2.3 get_paths()	. 88
4.37 src/exec/get_cmd_type.c File Reference	. 89
4.37.1 Detailed Description	. 89
4.37.2 Function Documentation	. 89
4.37.2.1 get_cmd_type()	. 89
4.38 src/exec/is_built_command.c File Reference	. 90
4.38.1 Function Documentation	. 90
4.38.1.1 is_built_command()	. 90
4.39 src/exec/manage_exec_data.c File Reference	. 90
4.39.1 Detailed Description	. 91
4.39.2 Function Documentation	. 91
4.39.2.1 free_exec_data()	. 91
4.39.2.2 init_exec_data()	. 91
4.39.2.3 manage_saved_fd()	. 91
4.40 src/format/check_exception.c File Reference	. 92
4.40.1 Function Documentation	. 92
4.40.1.1 handle_exceptions_and_quotes()	. 92
4.40.1.2 handle_skipping()	. 93
4.40.1.3 is_an_exception()	. 93
4.40.1.4 is_separator_or_redirection()	. 93
4.41 src/format/format.c File Reference	. 94
4.41.1 Detailed Description	. 94
4.41.2 Function Documentation	. 94
4.41.2.1 check_quote()	. 94
4.41.2.2 format_input()	. 95
4.42 src/format/put_spaces.c File Reference	. 95
4.42.1 Detailed Description	. 95
4.42.2 Function Documentation	. 96
4.42.2.1 add_spaces_around_operators()	. 96
4.43 src/alobhings/get_alobhings.c.File_Reference	96

4.43.1 Detailed Description
4.43.2 Function Documentation
4.43.2.1 create_concat_array()
4.43.2.2 get_globbings()
4.44 src/globbings/is_gobbling.c File Reference
4.44.1 Detailed Description
4.44.2 Function Documentation
4.44.2.1 is_gobbling()
4.45 src/input/autocompletion.c File Reference
4.45.1 Detailed Description
4.45.2 Function Documentation
4.45.2.1 autocomplete_tabs()
4.46 src/input/check_shortcut.c File Reference
4.46.1 Detailed Description
4.46.2 Function Documentation
4.46.2.1 check_shortcut()
4.47 src/input/get_input.c File Reference
4.47.1 Detailed Description
4.47.2 Function Documentation
4.47.2.1 display_line()
4.47.2.2 get_input()
4.48 src/input/handle_sequences.c File Reference
4.48.1 Detailed Description
4.48.2 Function Documentation
4.48.2.1 handle_sequences()
4.49 src/input/manage_input.c File Reference
4.49.1 Detailed Description
4.49.2 Function Documentation
4.49.2.1 manage_input()
4.50 src/input/manage_input_struct.c File Reference
4.50.1 Detailed Description
4.50.2 Function Documentation
4.50.2.1 free_input()
4.50.2.2 init_input()
4.51 src/parsing/command_handling/open_fd_redirections.c File Reference
4.51.1 Detailed Description
4.51.2 Function Documentation
4.51.2.1 open_input_fd()
4.51.2.2 open_output_fd()
4.52 src/parsing/command_handling/separators.c File Reference
4.52.1 Detailed Description
4.52.2 Function Documentation

4.52.2.1 change_separator()	105
4.52.2.2 get_highest_separator()	106
4.52.2.3 is_double_separator()	106
4.52.2.4 is_simple_separator()	107
4.53 src/parsing/command_handling/simple_command.c File Reference	108
4.53.1 Detailed Description	108
4.53.2 Function Documentation	108
4.53.2.1 add_command()	108
4.53.2.2 check_input_redirection()	109
4.53.2.3 check_output_redirection()	109
4.53.2.4 create_command_node()	110
4.53.2.5 handle_no_separator_command()	110
4.54 src/parsing/is_command_valid/check_double_and.c File Reference	110
4.54.1 Detailed Description	110
4.54.2 Function Documentation	111
4.54.2.1 is_valid_and_syntax()	111
4.55 src/parsing/is_command_valid/check_matched.c File Reference	111
4.55.1 Detailed Description	111
4.55.2 Function Documentation	111
4.55.2.1 is_nesting_matched()	111
4.56 src/parsing/is_command_valid/check_pipes.c File Reference	112
4.56.1 Detailed Description	112
4.56.2 Function Documentation	112
4.56.2.1 is_null_command()	112
4.56.2.2 is_space()	113
4.56.2.3 is_valid_pipe_syntax()	114
4.56.2.4 update_quote_state()	114
4.57 src/parsing/is_command_valid/check_semicolons.c File Reference	115
4.57.1 Detailed Description	115
4.57.2 Function Documentation	115
4.57.2.1 is_valid_semicolon_syntax()	115
4.58 src/parsing/is_command_valid/is_command_valid.c File Reference	115
4.58.1 Detailed Description	116
4.58.2 Function Documentation	116
4.58.2.1 is_valid_syntax()	116
4.59 src/parsing/main_parsing/nesting_handling.c File Reference	116
4.59.1 Function Documentation	116
4.59.1.1 change_backtick_and_quotes()	116
4.59.1.2 change_depth()	117
4.59.1.3 cut_parenthesis()	117
4.59.1.4 is_command_inside_parenthesis()	117
4.59.1.5 is_nested()	118

4.60 src/parsing/main_parsing/parser.c File Reference	8
4.60.1 Detailed Description	8
4.60.2 Function Documentation	8
4.60.2.1 choose_parsing()	8
4.60.2.2 cut_substring()	9
4.60.2.3 get_number_subcommands()	9
4.60.2.4 parse_commands()	0
4.60.2.5 parse_line()	0
4.61 src/utilities/free_shell.c File Reference	0
4.61.1 Detailed Description	1
4.61.2 Function Documentation	1
4.61.2.1 free_aliases()	1
4.61.2.2 free_commands()	1
4.61.2.3 free_shell()	1
4.62 src/utilities/init_shell.c File Reference	2
4.62.1 Detailed Description	2
4.62.2 Function Documentation	2
4.62.2.1 init_struct()	2
4.63 src/utilities/int_to_str.c File Reference	3
4.63.1 Function Documentation	3
4.63.1.1 int_to_str()	3
4.64 src/utilities/manage_char_array.c File Reference	3
4.64.1 Detailed Description	4
4.64.2 Function Documentation	4
4.64.2.1 del_elt()	4
4.64.2.2 dup_array()	4
4.64.2.3 free_array()	4
4.65 src/utilities/my_str_to_word_array.c File Reference	5
4.65.1 Detailed Description	5
4.65.2 Function Documentation	5
4.65.2.1 my_str_to_word_array()	5
4.66 src/utilities/str_alphanum.c File Reference	6
4.66.1 Detailed Description	6
4.66.2 Function Documentation	6
4.66.2.1 str_isalphanum()	6
4.67 src/exec/strjoin.c File Reference	6
4.67.1 Detailed Description	7
4.67.2 Function Documentation	7
4.67.2.1 strjoin()	7
4.68 src/utilities/strjoin.c File Reference	
4.68.1 Detailed Description	8
4.68.2 Function Documentation	8

	4.68.2.1 strjoin()	 	 	 	 	 		 		 128
Index										129

Chapter 1

Class Index

1.1 Class List

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

aliases_s .													 					 								7
commands_s													 					 								8
env_s													 					 								9
errno_outputs													 					 								9
exec_data_s																		 								10
input_s																		 								11
parsing_state_																										
shell_s																		 								13
tab builtins t				_	_		 		_				 		_		_	 				_				14

2 Class Index

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

include/builtins.h	17
include/error_outputs.h	28
include/exec.h	29
include/format.h	35
include/globbings.h	40
include/input.h	42
include/meowsh.h	47
include/parser.h	50
include/utilities.h	64
src/builtins/alias/alias.c	
All functions to handle the adding of aliases	68
src/builtins/alias/handle_alias.c	
Functions to manage aliases in the shell	69
src/builtins/alias/unalias.c	
Functions to handle the deletion of aliases in the shell	71
src/builtins/cd/cd_tilde.c	
Functions to handle the tilde (\sim) for the cd command	71
src/builtins/cd/get_home_user.c	
Functions to retrieve a user's home directory	72
src/builtins/cd/my_cd.c	
Implementation of the cd command for the shell $\ldots\ldots\ldots\ldots\ldots\ldots$	73
src/builtins/cd/my_getenv.c	
Retrieves the value of an environment variable	74
src/builtins/env/env.c	
Contains the implementation of the env command	75
src/builtins/env/setenv.c	
Implementation of the setenv builtin command	75
src/builtins/env/unsetenv.c	
Functions to handle the unsetenv builtin command	78
src/builtins/exec/my_exit.c	
Implementation of the exit builtin command	79
src/builtins/history/handle_history.c	
Functions to manage the shell's command history	80
src/builtins/which/which.c	
Implementation of the which builtin command	82

File Index

src/builtins/which_and_where/where.c	
Implementation of the where builtin command	84
src/builtins/which_and_where/which.c	
Implementation of the which builtin command	83
src/exec/check_signal.c	
Functions to handle and display process signals	85
src/exec/exec_cmd.c	06
Implementation of command execution for the 42sh shell	86
Functions to locate the executable path of a command	87
src/exec/get_cmd_type.c	07
Determines the type of a command in the 42sh shell	89
src/exec/is_built_command.c	90
src/exec/manage_exec_data.c	
Manages the initialization, cleanup, and handling of execution data	90
src/exec/strjoin.c	
Function to concatenate two strings	126
src/format/check_exception.c	92
src/format/format.c	
Functions to format and process shell input	94
src/format/put_spaces.c	
Functions to add spaces around operators in a shell input string	95
src/globbings/get_globbings.c	
Functions for retrieving and managing globbing results	96
src/globbings/is_gobbling.c	07
Function to check for globbing patterns in a string	97
src/input/autocompletion.c	00
Implementation of the tab-autocompletion feature	98
Handles keyboard shortcuts for cursor movement and screen clearing	99
src/input/get_input.c	99
\cdot \cdot \cdot \cdot \cdot	100
src/input/handle_sequences.c	
	101
src/input/manage_input.c	
\cdot	102
src/input/manage_input_struct.c	
Functions to initialize and free the input structure	102
src/parsing/command_handling/open_fd_redirections.c	
' '	104
src/parsing/command_handling/separators.c	
	105
src/parsing/command_handling/simple_command.c	
	108
src/parsing/is_command_valid/check_double_and.c	440
, ,	110
src/parsing/is_command_valid/check_matched.c	444
· · · · · · · · · · · · · · · · · · ·	111
src/parsing/is_command_valid/check_pipes.c Functions to check the validity of pipes' syntax in the line	112
src/parsing/is_command_valid/check_semicolons.c	112
	115
src/parsing/is_command_valid/is_command_valid.c	
	115
····	116
src/parsing/main_parsing/parser.c	
Main parsing functions that recursively parse into subcommands	118

2.1 File List 5

src/utilities/free_shell.c	
Functions to free memory allocated for the shell structure	20
src/utilities/init_shell.c	
Initialization functions for the shell program	22
src/utilities/int_to_str.c	23
src/utilities/manage_char_array.c	
Utility functions to manage arrays of strings	23
src/utilities/my_str_to_word_array.c	
Functions to split a string into an array of words	25
src/utilities/str_alphanum.c	
Utility function to check if a string is alphanumeric	26
src/utilities/strjoin.c	
Function to concatenate two strings	27

6 File Index

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

```
\verb"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

 $\verb|bool parsing_state_s::in_double_quote|\\$

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)

18 File Documentation

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 (\sim) 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()

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()

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

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

4.1.2.3 add_to_history()

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()

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.

20 File Documentation

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()

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()

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()

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()

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()

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()

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.

22 File Documentation

4.1.2.11 get_value_key()

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()

Handles tilde (\sim) and other special cases for the cd command.

This function processes the input directory string and handles:

- ullet \sim for the current user's home directory.
- \sim 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()

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()

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()

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()

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()

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()

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()

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()

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()

Handles the exit command.

This function processes the <code>exit</code> 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 <code>exit_status</code> is used.

Parameters

shell	Pointer to the shell structure.
command	Array of strings representing the <code>exit</code> 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()

Handles the seteny builtin command.

This function implements the logic for the seteny 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()

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.

4.2 builtins.h

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
           #include "meowsh.h"
#include "utilities.h"
00010
00011
00012
           #include <errno.h>
00013
           #include <sys/stat.h>
           #include <fcntl.h>
00014
           #define NB_BUILT_IN 12
00015
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);
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,
           __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,
           __attribute_maybe_unused__ int local);
00059
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."},
{ENOTDIR, "Not a directory."},
{ENOTDIR, "Not a directory."},
{EPERM, "Operation not permitted."},
{ETXTBSY, "Text file busy."},
{-1, NULL}
```

4.4 error_outputs.h

4.4 error_outputs.h

Go to the documentation of this file.

```
00002 ** EPITECH PROJECT, 2024
00003 ** error ouptus tab
00004 ** File description:
00005 ** my_constants definition
00006 */
00007
00008 #ifndef MY_CONSTANTS
            #define MY_CONSTANTS
00009
00010
                #include <errno.h>
                #include <stdlib.h>
00011
00012
00013 typedef struct errno_outputs {
            int errnum;
const char *errmsg;
00015
00016 } errno_outputs_t;
00017
00018 errno_outputs_t const errno_table[] = {
            Errno_outputs_t const errno_table[] = {
    {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 "}
00021
00022
00023
00024
00025
00027
                 {EMFILE, "Too many open files."},
00028
                  {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."},
00029
00030
00031
                 ENOTDIR, "Not a directory.",
{EPERM, "Operation not permitted."},
{ETXTBSY, "Text file busy."},
00033
00034
00035
                  \{-1, NULL\}
00036
00037 };
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()

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()

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()

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()

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()

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()

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()

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()

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()

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.

4.6 exec.h 35

Parameters

saved

Array containing the saved file descriptors for stdin and stdout.

4.6 exec.h

Go to the documentation of this file.

```
00002 ** EPITECH PROJECT, 2024
00003 ** 42sh
00004 ** File description:
00005 ** exec.h
00006 */
00007
00008 #ifndef EXEC_H
00009
            #define EXEC_H
00010
             #include "meowsh.h"
00011
             #include "builtins.h"
00012
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
{"setenv", &my_setenv, 0},
00027
              {"env", &my_env, 0},
              {"unsetenv", &my_unsetenv, 0},
{"history", &display_history, 0},
{"alias", &interpret_alias, 0},
{"unalias", &interpret_unalias, 0},
00028
00029
00030
00031
             {"unalias", &interpret_unalias,
{"exit", &my_exit, 0},
{"set", &my_setenv, 1},
{"unset", &my_unsetenv, 1},
{"which", &interpret_which, 0},
{"where", &interpret_where, 0}
00032
00033
00034
00035
00036
00037 };
00038
00039 typedef struct exec_data_s {
          int nb_elt;
int **pipe;
00040
00041
00042
             int *pid;
00043
           int *status;
int *cmd_type;
00044
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]);
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()

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()

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()

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()

Handles exceptions and quotes for a single argument.

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

4.8 format.h 39

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()

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
       #define FORMAT
00009
          #include "meowsh.h"
00010
          #include <stdint.h>
00011
00012
          #include <stddef.h>
00013
          #include <stdlib.h>
00014
          #include <string.h>
          #include <stdio.h>
#define NB_EXCEPTIONS 4
00015
00016
00017
          #define MAX_NBR 11
00019 static const char *exceptions[NB_EXCEPTIONS] __attribute__((unused)) = {
```

```
"alias",
00021
           "unalias",
           "which",
00022
           "where"
00023
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,
           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()

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()

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()

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_
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()

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()

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()

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()

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()

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()

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	t 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()

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()

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 47

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
            #define INPUT_H
00009
00010
00011
            #include <termios.h>
00012
            #include <unistd.h>
00013
            #include <stdio.h>
00014
            #include <stdlib.h>
            #include <string.h>
00015
            #include <ctype.h>
00016
            #include "meowsh.h"
            #define BLUE "\033[1;34m"
#define RESET "\033[0m"
#define PURPLE "\033[0;38;5;56m"
00018
00019
00020
00021
00022 typedef struct input_s {
00023
            char *str_input;
00024
            char char_input;
00025
            size_t input_len;
00026
            size_t cursor_pos;
           int history_pos;
char *stock_history;
00027
00028
00029 } input_t;
00030
00031 int handle_sequences(input_t *input, shell_t *shell);
00032 int manage_input(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	

4.14 meowsh.h 49

4.13.3 Function Documentation

4.13.3.1 str_isalphanum()

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.

```
00002 ** EPITECH PROJECT, 2025
00003 ** meowsh
00004 ** File description:
00005 ** Main header file
00006 */
00008 #ifndef MEOWSH_H
       #define MEOWSH_H
00009
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,
          COLUMN,
00022
00023
         AND OR
00024 } sep_t;
00025
00027 typedef struct env_s {
        char **local;
00028
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;
         int fd_input;
00048
00049
         int fd_output;
00050
         char *heredoc;
```

```
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;
           env_t *env;
00059
          aliases_t **aliases;
         char **history;
int history_size;
char *owd;
int exit_status;
bool run;
00060
00061
00062
00063
00064
00065 } shell_t;
00066
00068
00070 typedef struct {
00071 char *command;
00072 int (*builtin_
           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()

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()

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()

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()

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()

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()

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()

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()

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()

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()

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()

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()

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()

Check if the command is entirely wrapped in parentheses.

Parameters

ne The input command line.	line
----------------------------	------

Returns

1 if command is parenthesized, 0 otherwise.

4.15.2.14 is_double_separator()

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()

Check if the parser is currently in a nested state.

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

Parameters

Returns

1 if nested, 0 otherwise.

4.15.2.16 is_nesting_matched()

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

Parameters

Returns

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

4.15.2.17 is_null_command()

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()

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 ( {\tt 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()

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()

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()

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()

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()

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()

Opens the output file descriptor for a command.

Parameters

cmd	md Pointer to the command structure (commands_t).	
truncate If true, the file is truncated; if false, data is append		
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()

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()

Entry point for parsing a command line.

Validates the syntax and initializes the command tree.

4.16 parser.h 63

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()

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

С	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
       #define PARSER_H
#include "meowsh.h"
#include <string.h>
00009
00010
00011
00012
          #include <fcntl.h>
          #include "utilities.h"
00014
00015 // Structs
00016
00017 typedef struct parsing_state_s {
00018
          char *line;
sep_t separator;
00019
00020
           int current_depth;
00021
          bool in_single_quote;
00022
          bool in_double_quote;
         bool in_backtick;
int start_index;
00023
00024
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,
          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,
        sep_t separator, bool does_increment);
00044
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);
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()

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()

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()

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()

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()

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()

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()

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).
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()

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()

```
\label{eq:char * strjoin (} \operatorname{char} * str1, \\ \operatorname{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>
#include "meowsh.h"
00014
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()

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()

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()

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()

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

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()

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 (\sim) 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 (\sim) for the cd command.

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

4.22.2 Function Documentation

4.22.2.1 handle_tilde()

Handles tilde (\sim) and other special cases for the cd command.

This function processes the input directory string and handles:

- \sim 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()

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 ${\tt cd}$ 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()

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()

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()

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 seteny command, which is used to set or modify environment variables in the shell.

4.27.2 Function Documentation

4.27.2.1 add env()

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()

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()

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()

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()

Handles the setenv builtin command.

This function implements the logic for the seteny 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()

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()

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()

Handles the exit command.

This function processes the <code>exit</code> 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 <code>exit_status</code> is used.

Parameters

shell	Pointer to the shell structure.
command	Array of strings representing the <code>exit</code> 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()

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()

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()

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()

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()

Load the history into the shell.

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

Parameters

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()

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	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()

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()

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()

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 comm	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()

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()

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()

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()

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()

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()

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

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()

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:

- ${\tt NOTHING}$ if the command is NULL or empty.
- SUBCMD if the command contains subcommands.
- BUILT_CMD if the command is a built-in command.
- $\ensuremath{\mathsf{CMD}}$ 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()

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()

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

Pointer to the execution data structure.
--

4.39.2.2 init_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()

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.	1

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()

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()

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()

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.
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()

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()

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()

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.		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()

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

<i>input</i> 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()

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

Returns

Pointer to the concatenated string, or NULL on failure.

4.43.2.2 get_globbings()

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()

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()

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()

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)

char * get_input (shell_t *shell)

Reads and processes user input in raw mode.

Displays the current input line in the terminal.

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()

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()

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()

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()

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()

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()

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.	
Ì	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()

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()

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()

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()

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()

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()

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()

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()

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()

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()

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()

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()

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()

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

Parameters

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()

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 ( {\tt 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()

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()

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

С	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()

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()

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()

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()

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()

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()

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()

Check if the parser is currently in a nested state.

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

Parameters

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()

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()

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()

Get the number of subcommands in a command.

Counts the number of subcommands using a separator.

Parameters

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

Returns

The number of subcommands.

4.60.2.4 parse_commands()

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()

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()

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()

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()

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()

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()

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()

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()

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()

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()

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 ( {\tt 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

 $\ensuremath{\mathbf{1}}$ if the string is alphanumeric, $\ensuremath{\mathbf{0}}$ 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()

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()

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

add_command	add_node, 19
parser.h, 52	add_to_history, 19
simple_command.c, 108	BUFFER_SIZE, 18
add_elt	delete_node, 19
builtins.h, 18	display_history, 20
unsetenv.c, 78	exec_history, 20
add_env	free_history, 20
setenv.c, 76	free_node, 21
add node	get_history, 21
builtins.h, 19	get_root_home, 21
handle_alias.c, 70	get_value_key, 21
add_spaces_around_operators	handle_tilde, 22
format.h, 36	interpret_alias, 22
put_spaces.c, 96	interpret_unalias, 23
add_to_history	interpret_where, 23
builtins.h, 19	interpret_which, 23
handle history.c, 80	is_a_builtin_command_where, 24
alias.c	is a builtin command which, 24
interpret_alias, 69	my_cd, 25
aliases	my_env, 25
shell_s, 13	my_exit, 25
aliases_s, 7	my_setenv, 26
browsed, 7	my_unsetenv, 26
new_name, 7	NB BUILT IN, 18
next, 7	115_55121_111, 10
original, 7	cd_tilde.c
aliases t	handle_tilde, 72
meowsh.h, 48	change_backtick_and_quotes
AND	nesting_handling.c, 116
meowsh.h, 48	parser.h, 52
AND OR	change_depth
meowsh.h, 48	nesting_handling.c, 117
autocomplete_tabs	parser.h, 52
autocompletion.c, 98	change_env
input.h, 43	setenv.c, 76
autocompletion.c	change_separator
autocomplete_tabs, 98	parser.h, 52
aatooop.oto_tabo, oo	separators.c, 105
BLUE	char_input
input.h, 43	input_s, 11
browsed	check_double_and.c
aliases_s, 7	is_valid_and_syntax, 111
BUFFER_SIZE	check_exception.c
builtins.h, 18	handle_exceptions_and_quotes, 92
BUILT CMD	handle_skipping, 92
exec.h, 30	is_an_exception, 93
builtin_func	is_separator_or_redirection, 93
tab_builtins_t, 14	check_input_redirection
builtins.h	parser.h, 54
add_elt, 18	simple_command.c, 109
- '	. –

check_matched.c	globbings.h, 40
is_nesting_matched, 111	current_depth
check_output_redirection	parsing_state_s, 12
parser.h, 54	current_subcommand_index
simple_command.c, 109	parsing_state_s, 12
check_path	cursor_pos
exec.h, 31	input_s, 11
get_cmd_path.c, 88	cut_parenthesis
check_pipes.c	nesting_handling.c, 117
is_null_command, 112	parser.h, 55
is_space, 112	cut_substring
is_valid_pipe_syntax, 114	parser.c, 119
update_quote_state, 114	
check_quote	del_elt
format.c, 94	manage_char_array.c, 124
format.h, 37	utilities.h, 65
check_semicolons.c	delete_node
is valid semicolon syntax, 115	builtins.h, 19
check shortcut	handle_alias.c, 70
check_shortcut.c, 99	depth
input.h, 44	commands_s, 8
check_shortcut.c	display_history
check_shortcut, 99	builtins.h, 20
check_signal	handle_history.c, 81
check_signal.c, 86	display_line
exec.h, 31	get_input.c, 100
check_signal.c	input.h, 44
check_signal, 86	dup_array
choose_parsing	manage_char_array.c, 124
parser c. 118	utilities.h, 65
parser.c, 118 parser.h. 54	utilities.h, 65
parser.h, 54	utilities.h, 65 env
parser.h, 54 CMD	
parser.h, 54 CMD exec.h, 30	env
parser.h, 54 CMD exec.h, 30 cmd_type	env shell_s, 13
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10	env shell_s, 13 env.c
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10 COLUMN	env shell_s, 13 env.c my_env, 75
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10 COLUMN meowsh.h, 48	env shell_s, 13 env.c my_env, 75 env_s, 9
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10 COLUMN meowsh.h, 48 command	env shell_s, 13 env.c my_env, 75 env_s, 9 global, 9
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10 COLUMN meowsh.h, 48 command commands_s, 8	env shell_s, 13 env.c my_env, 75 env_s, 9 global, 9 local, 9
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10 COLUMN meowsh.h, 48 command commands_s, 8 tab_builtins_t, 14	env shell_s, 13 env.c my_env, 75 env_s, 9 global, 9 local, 9 env_t
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10 COLUMN meowsh.h, 48 command commands_s, 8 tab_builtins_t, 14 commands	env shell_s, 13 env.c my_env, 75 env_s, 9 global, 9 local, 9 env_t meowsh.h, 48
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10 COLUMN meowsh.h, 48 command commands_s, 8 tab_builtins_t, 14 commands shell_s, 13	env shell_s, 13 env.c my_env, 75 env_s, 9 global, 9 local, 9 env_t meowsh.h, 48 errmsg
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10 COLUMN meowsh.h, 48 command commands_s, 8 tab_builtins_t, 14 commands shell_s, 13 commands_s, 8	env shell_s, 13 env.c my_env, 75 env_s, 9 global, 9 local, 9 env_t meowsh.h, 48 errmsg errno_outputs, 10
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10 COLUMN meowsh.h, 48 command commands_s, 8 tab_builtins_t, 14 commands shell_s, 13 commands_s, 8 command, 8	env shell_s, 13 env.c my_env, 75 env_s, 9 global, 9 local, 9 env_t meowsh.h, 48 errmsg errno_outputs, 10 errno_outputs, 9
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10 COLUMN meowsh.h, 48 command commands_s, 8 tab_builtins_t, 14 commands shell_s, 13 commands_s, 8 commands_s, 8 depth, 8	env shell_s, 13 env.c my_env, 75 env_s, 9 global, 9 local, 9 env_t meowsh.h, 48 errmsg errno_outputs, 10 errno_outputs, 9 errmsg, 10
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10 COLUMN meowsh.h, 48 command commands_s, 8 tab_builtins_t, 14 commands shell_s, 13 commands_s, 8 command, 8 depth, 8 fd_input, 8	env shell_s, 13 env.c my_env, 75 env_s, 9 global, 9 local, 9 env_t meowsh.h, 48 errmsg errno_outputs, 10 errno_outputs, 9 errmsg, 10 errnum, 10
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10 COLUMN meowsh.h, 48 command commands_s, 8 tab_builtins_t, 14 commands shell_s, 13 commands_s, 8 command, 8 depth, 8 fd_input, 8 fd_output, 8	env shell_s, 13 env.c my_env, 75 env_s, 9 global, 9 local, 9 env_t meowsh.h, 48 errmsg errno_outputs, 10 errno_outputs, 9 errmsg, 10 errnum, 10 errno_outputs_t
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10 COLUMN meowsh.h, 48 command commands_s, 8 tab_builtins_t, 14 commands shell_s, 13 commands_s, 8 command, 8 depth, 8 fd_input, 8 fd_output, 8 heredoc, 8	env shell_s, 13 env.c my_env, 75 env_s, 9 global, 9 local, 9 env_t meowsh.h, 48 errmsg errno_outputs, 10 errno_outputs, 9 errmsg, 10 errnum, 10 errno_outputs_t error_outputs.h, 28
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10 COLUMN meowsh.h, 48 command commands_s, 8 tab_builtins_t, 14 commands shell_s, 13 commands_s, 8 commands_s, 8 depth, 8 fd_input, 8 fd_output, 8 heredoc, 8 path, 8	env shell_s, 13 env.c my_env, 75 env_s, 9 global, 9 local, 9 env_t meowsh.h, 48 errmsg errno_outputs, 10 errno_outputs, 9 errmsg, 10 errnum, 10 errno_outputs_t error_outputs.h, 28 errno_table
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10 COLUMN meowsh.h, 48 command commands_s, 8 tab_builtins_t, 14 commands shell_s, 13 commands_s, 8 command, 8 depth, 8 fd_input, 8 fd_output, 8 heredoc, 8 path, 8 separator, 8	env shell_s, 13 env.c my_env, 75 env_s, 9 global, 9 local, 9 env_t meowsh.h, 48 errmsg errno_outputs, 10 errno_outputs, 9 errmsg, 10 errnum, 10 errno_outputs_t error_outputs.h, 28 errno_table error_outputs.h, 28
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10 COLUMN meowsh.h, 48 command commands_s, 8 tab_builtins_t, 14 commands shell_s, 13 commands_s, 8 command, 8 depth, 8 fd_input, 8 fd_output, 8 heredoc, 8 path, 8 separator, 8 subcommands, 8	env shell_s, 13 env.c my_env, 75 env_s, 9 global, 9 local, 9 env_t meowsh.h, 48 errmsg errno_outputs, 10 errno_outputs, 9 errmsg, 10 errnum, 10 errno_outputs_t error_outputs.h, 28 errno_table error_outputs.h, 28 errnum
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10 COLUMN meowsh.h, 48 command commands_s, 8 tab_builtins_t, 14 commands shell_s, 13 commands_s, 8 command, 8 depth, 8 fd_input, 8 fd_output, 8 heredoc, 8 path, 8 separator, 8 subcommands_t	env shell_s, 13 env.c my_env, 75 env_s, 9 global, 9 local, 9 env_t meowsh.h, 48 errmsg errno_outputs, 10 errno_outputs, 9 errmsg, 10 errnum, 10 errno_outputs_t error_outputs.h, 28 errno_table error_outputs.h, 28 errnum errno_outputs, 10
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10 COLUMN meowsh.h, 48 command commands_s, 8 tab_builtins_t, 14 commands shell_s, 13 commands_s, 8 command, 8 depth, 8 fd_input, 8 fd_output, 8 heredoc, 8 path, 8 separator, 8 subcommands_t meowsh.h, 48	env shell_s, 13 env.c my_env, 75 env_s, 9 global, 9 local, 9 env_t meowsh.h, 48 errmsg errno_outputs, 10 errno_outputs, 9 errmsg, 10 errnum, 10 errno_outputs_t error_outputs.h, 28 errno_table error_outputs.h, 28 errnum errno_outputs, 10 error_outputs.h
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10 COLUMN meowsh.h, 48 command commands_s, 8 tab_builtins_t, 14 commands shell_s, 13 commands_s, 8 command, 8 depth, 8 fd_input, 8 fd_output, 8 heredoc, 8 path, 8 separator, 8 subcommands_t meowsh.h, 48 create_command_node	env shell_s, 13 env.c my_env, 75 env_s, 9 global, 9 local, 9 env_t meowsh.h, 48 errmsg errno_outputs, 10 errno_outputs, 9 errmsg, 10 errnum, 10 errno_outputs_t error_outputs.h, 28 errno_table error_outputs.h, 28 errnum errno_outputs.h errno_outputs.h errno_outputs.h errno_outputs_t, 28
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10 COLUMN meowsh.h, 48 command commands_s, 8 tab_builtins_t, 14 commands shell_s, 13 commands_s, 8 command, 8 depth, 8 fd_input, 8 fd_output, 8 heredoc, 8 path, 8 separator, 8 subcommands, 8 commands_t meowsh.h, 48 create_command_node parser.h, 55	env shell_s, 13 env.c my_env, 75 env_s, 9 global, 9 local, 9 env_t meowsh.h, 48 errmsg errno_outputs, 10 errno_outputs, 9 errmsg, 10 errnum, 10 errno_outputs_t error_outputs.h, 28 errno_table error_outputs.h, 28 errnum errno_outputs.h errno_outputs, 10 error_outputs.h errno_outputs, 28 errnum errno_outputs, 10
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10 COLUMN meowsh.h, 48 command commands_s, 8 tab_builtins_t, 14 commands shell_s, 13 commands_s, 8 command, 8 depth, 8 fd_input, 8 fd_output, 8 heredoc, 8 path, 8 separator, 8 subcommands_t meowsh.h, 48 create_command_node parser.h, 55 simple_command.c, 109	env shell_s, 13 env.c my_env, 75 env_s, 9 global, 9 local, 9 env_t meowsh.h, 48 errmsg errno_outputs, 10 errno_outputs, 9 errmsg, 10 errnum, 10 errno_outputs_t error_outputs.h, 28 errno_table error_outputs.h, 28 errnum errno_outputs.h 10 error_outputs.h 28 errnum errno_outputs.h 28 errnum errno_outputs.h 28 errnum errno_outputs.h 28 errnum errno_outputs.h 28 errno_table, 28 exec.h
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10 COLUMN meowsh.h, 48 command commands_s, 8 tab_builtins_t, 14 commands shell_s, 13 commands_s, 8 command, 8 depth, 8 fd_input, 8 fd_output, 8 heredoc, 8 path, 8 separator, 8 subcommands_t meowsh.h, 48 create_command_node parser.h, 55 simple_command.c, 109 create_concat_array	env shell_s, 13 env.c my_env, 75 env_s, 9 global, 9 local, 9 env_t meowsh.h, 48 errmsg errno_outputs, 10 errno_outputs, 9 errmsg, 10 errnum, 10 errno_outputs_t error_outputs.h, 28 errno_table error_outputs.h, 28 errnum errno_outputs, 10 error_outputs.h errno_outputs, 10 error_outputs.h errno_outputs, 28 errno_table, 28 exec.h BUILT_CMD, 30 check_path, 31
parser.h, 54 CMD exec.h, 30 cmd_type exec_data_s, 10 COLUMN meowsh.h, 48 command commands_s, 8 tab_builtins_t, 14 commands shell_s, 13 commands_s, 8 command, 8 depth, 8 fd_input, 8 fd_output, 8 heredoc, 8 path, 8 separator, 8 subcommands_t meowsh.h, 48 create_command_node parser.h, 55 simple_command.c, 109	env shell_s, 13 env.c my_env, 75 env_s, 9 global, 9 local, 9 env_t meowsh.h, 48 errmsg errno_outputs, 10 errno_outputs, 9 errmsg, 10 errnum, 10 errno_outputs_t error_outputs.h, 28 errno_table error_outputs.h, 28 errnum errno_outputs, 10 error_outputs.h errno_outputs, 10 error_outputs.h errno_outputs, 28 errnum errno_outputs, 28 errno_table, 28 exec.h BUILT_CMD, 30

exec_cmd, 31	free_history
exec_data_t, 31	builtins.h, 20
free_exec_data, 32	handle_history.c, 81
get_cmd_path, 32	free_input
get_cmd_type, 32	input.h, 45
get_paths, 33	manage_input_struct.c, 103
init_exec_data, 33	free_node
manage_saved_fd, 33	builtins.h, 21
NOTHING, 30	handle_alias.c, 70
SUBCMD, 30	free_shell
exec_cmd	free_shell.c, 121
exec.h, 31	utilities.h, 66
exec_cmd.c, 87	free_shell.c
exec_cmd.c	free_aliases, 121
exec_cmd, 87	free_commands, 121
exec_data_s, 10	free_shell, 121
cmd_type, 10	get and noth
nb_elt, 10	get_cmd_path
pid, 10	exec.h, 32
pipe, 10	get_cmd_path.c, 88
prev_depth, 10	get_cmd_path.c
status, 11	check_path, 88
exec_data_t	get_cmd_path, 88
exec.h, 31	get_paths, 88
exec_history	get_cmd_type
builtins.h, 20	exec.h, 32
handle_history.c, 81	get_cmd_type.c, 89
exit_status	get_cmd_type.c
shell_s, 14	get_cmd_type, 89
fd_input	get_globbings
_ ·	get_globbings.c, 97 globbings.h, 41
commands_s, 8 fd_output	get globbings.c
commands_s, 8	create_concat_array, 96
format.c	get_globbings, 97
check_quote, 94	get_highest_separator
format_input, 95	parser.h, 55
format.h	separators.c, 106
add_spaces_around_operators, 36	get_history
check quote, 37	builtins.h, 21
format_input, 37	handle history.c, 82
handle_exceptions_and_quotes, 37	get_home_user.c
handle_skipping, 39	get_root_home, 73
MAX NBR, 36	get_not_nome, 75
NB_EXCEPTIONS, 36	get_input.c, 100
format_input	input.h, 45
format.c, 95	get_input.c
format.h, 37	display line, 100
free aliases	get_input, 100
free_shell.c, 121	get_line_env
free_array	setenv.c, 77
manage_char_array.c, 124	get_number_subcommands
utilities.h, 65	parser.c, 119
free_commands	parser.h, 56
free_shell.c, 121	get_paths
utilities.h, 66	exec.h, 33
free_exec_data	
exec.h, 32	get_cmd_path.c, 88
manage_exec_data.c, 91	get_root_home builtins.h, 21
manage_exec_uata.c, 31	Dunini3.11, 2.1

get_home_user.c, 73	include/input.h, 42, 47
get_value_key	include/meowsh.h, 47, 49
builtins.h, 21	include/parser.h, 50, 63
my_getenv.c, 74	include/utilities.h, 64, 68
global	init_exec_data
env s, 9	exec.h, 33
globbings.h	manage_exec_data.c, 91
create_concat_array, 40	init_input
·	_ ·
get_globbings, 41	input.h, 46
is_gobbling, 41	manage_input_struct.c, 103
MAX_GLOBS, 40	init_shell.c
1 H P	init_struct, 122
handle_alias.c	init_struct
add_node, 70	init_shell.c, 122
delete_node, 70	utilities.h, 66
free_node, 70	input.h
handle_exceptions_and_quotes	autocomplete_tabs, 43
check_exception.c, 92	BLUE, 43
format.h, 37	check_shortcut, 44
handle_history.c	display_line, 44
add_to_history, 80	free input, 45
display_history, 81	get input, 45
exec_history, 81	handle_sequences, 45
free_history, 81	init_input, 46
get_history, 82	_ ·
handle_no_separator_command	input_t, 43
parser.h, 56	manage_input, 46
simple_command.c, 110	PURPLE, 43
• —	RESET, 43
handle_sequences	input_len
handle_sequences.c, 101	input_s, 11
input.h, 45	input_s, 11
handle_sequences.c	char_input, 11
handle_sequences, 101	cursor_pos, 11
handle_skipping	history_pos, 11
check_exception.c, 92	input_len, 11
format.h, 39	stock_history, 11
handle_tilde	str_input, 12
builtins.h, 22	input t
cd_tilde.c, 72	input.h, 43
heredoc	int_to_str
commands_s, 8	int_to_str.c, 123
history	utilities.h, 66
shell_s, 14	int_to_str.c
history_pos	
input_s, 11	int_to_str, 123
	interpret_alias
history_size shell s, 14	alias.c, 69
Shell_S, 14	builtins.h, 22
in backtick	interpret_unalias
-	builtins.h, 23
parsing_state_s, 12	unalias.c, 71
in_double_quote	interpret_where
parsing_state_s, 12	builtins.h, 23
in_single_quote	where.c, 85
parsing_state_s, 12	interpret_which
include/builtins.h, 17, 27	builtins.h, 23
include/error_outputs.h, 28, 29	which.c, 82, 83
include/exec.h, 29, 35	is_a_builtin_command_where
include/format.h, 35, 39	builtins.h, 24
include/globbings.h, 40, 42	Junui J.H, 47

where.c, 85	tab_builtins_t, 15
is_a_builtin_command_which	
builtins.h, 24	manage_char_array.c
which.c, 84	del_elt, 124
is_a_good_var	dup_array, 124 free_array, 124
setenv.c, 77	manage_exec_data.c
is_an_exception	free_exec_data, 91
check_exception.c, 93	init_exec_data, 91
is_built_command	manage_saved_fd, 91
is_built_command.c, 90	manage input
is_built_command.c	input.h, 46
is_built_command, 90	manage_input.c, 102
is_command_inside_parenthesis	manage_input.c
nesting_handling.c, 117	manage_input, 102
parser.h, 56	manage_input_struct.c
is_command_valid.c	free_input, 103
is_valid_syntax, 116	init_input, 103
is_double_separator	manage saved fd
parser.h, 58	exec.h, 33
separators.c, 106	manage_exec_data.c, 91
is_gobbling	MAX GLOBS
globbings.h, 41	globbings.h, 40
is_gobbling.c, 97	MAX NBR
is_gobbling.c	format.h, 36
is_gobbling, 97	meowsh.h
is_nested	aliases t, 48
nesting_handling.c, 117	AND, 48
parser.h, 58	AND_OR, 48
is_nesting_matched	COLUMN, 48
check_matched.c, 111	commands t, 48
parser.h, 58	env t, 48
is_null_command	NONE, 48
check_pipes.c, 112	OR, 48
parser.h, 59 is_separator_or_redirection	PIPE, 48
check_exception.c, 93	sep_s, 48
_ ·	sep t, 48
is_simple_separator parser.h, 59	shell_t, 48
separators.c, 106	str isalphanum, 49
is_space	my cd
check_pipes.c, 112	builtins.h, 25
parser.h, 59	my_cd.c, 73
is_valid_and_syntax	my_cd.c
check_double_and.c, 111	my_cd, 73
parser.h, 60	my env
is_valid_pipe_syntax	builtins.h, 25
check_pipes.c, 114	env.c, 75
parser.h, 60	my_exit
is_valid_semicolon_syntax	builtins.h, 25
check_semicolons.c, 115	my_exit.c, 79
parser.h, 60	my exit.c
is_valid_syntax	my_exit, 79
is_command_valid.c, 116	my_getenv.c
parser.h, 61	get_value_key, 74
paroonii, or	my_setenv
line	builtins.h, 26
parsing_state_s, 13	setenv.c, 77
local	my_str_to_word_array
env_s, 9	my_str_to_word_array.c, 125
	. — — - •

utilities.h, 67	change_depth, 52
my_str_to_word_array.c	change_separator, 52
my_str_to_word_array, 125	check_input_redirection, 54
my_unsetenv	check_output_redirection, 54
builtins.h, 26	choose_parsing, 54
unsetenv.c, 78	create_command_node, 55
	cut_parenthesis, 55
NB_BUILT_IN	get_highest_separator, 55
builtins.h, 18	get_number_subcommands, 56
nb_elt	handle no separator command, 56
exec_data_s, 10	is_command_inside_parenthesis, 56
NB EXCEPTIONS	is_double_separator, 58
format.h, 36	is_nested, 58
nesting_handling.c	is_nesting_matched, 58
change_backtick_and_quotes, 116	_ - _
change_depth, 117	is_null_command, 59
cut_parenthesis, 117	is_simple_separator, 59
	is_space, 59
is_command_inside_parenthesis, 117	is_valid_and_syntax, 60
is_nested, 117	is_valid_pipe_syntax, 60
new_name	is_valid_semicolon_syntax, 60
aliases_s, 7	is_valid_syntax, 61
next	open_input_fd, 61
aliases_s, 7	open_output_fd, 62
NONE	parse_commands, 62
meowsh.h, 48	parse_line, 62
NOTHING	parsing_state_t, 51
exec.h, 30	update_quote_state, 63
	parsing_state_s, 12
open_fd_redirections.c	current_depth, 12
open_input_fd, 104	current_subcommand_index, 12
open_output_fd, 104	
open_input_fd	in_backtick, 12
open_fd_redirections.c, 104	in_double_quote, 12
parser.h, 61	in_single_quote, 12
open_output_fd	line, 13
open_fd_redirections.c, 104	separator, 13
parser.h, 62	start_index, 13
OR	parsing_state_t
	parser.h, 51
meowsh.h, 48	path
original	commands_s, 8
aliases_s, 7	pid
owd	exec_data_s, 10
shell_s, 14	PIPE
	meowsh.h, 48
parse_commands	pipe
parser.c, 119	exec data s, 10
parser.h, 62	prev_depth
parse_line	exec data s, 10
parser.c, 120	PURPLE
parser.h, 62	_
parser.c	input.h, 43
choose_parsing, 118	put_spaces.c
cut_substring, 119	add_spaces_around_operators, 96
get_number_subcommands, 119	DECET
parse_commands, 119	RESET
parse_line, 120	input.h, 43
parser.h	run
add_command, 52	shell_s, 14
change_backtick_and_quotes, 52	sep_s

meowsh.h, 48	src/format/put spaces.c, 95
sep_t	src/globbings/get_globbings.c, 96
meowsh.h, 48	src/globbings/is_gobbling.c, 97
separator	src/input/autocompletion.c, 98
commands_s, 8	src/input/check_shortcut.c, 99
parsing_state_s, 13	src/input/get_input.c, 100
separators.c	src/input/handle_sequences.c, 101
change_separator, 105	src/input/manage_input.c, 102
get_highest_separator, 106	src/input/manage input struct.c, 102
is_double_separator, 106	src/parsing/command handling/open fd redirections.c,
is_simple_separator, 106	104
setenv.c	src/parsing/command_handling/separators.c, 105
add_env, 76	src/parsing/command_handling/simple_command.c,
change_env, 76	108
get_line_env, 77	src/parsing/is_command_valid/check_double_and.c,
is_a_good_var, 77	110
my_setenv, 77	src/parsing/is_command_valid/check_matched.c, 111
shell_s, 13	src/parsing/is_command_valid/check_pipes.c, 112
aliases, 13	src/parsing/is_command_valid/check_semicolons.c,
commands, 13	
env, 13	src/parsing/is_command_valid/is_command_valid.c,
exit_status, 14	115
history, 14	src/parsing/main_parsing/nesting_handling.c, 116
history_size, 14	src/parsing/main_parsing/parser.c, 118
owd, 14	src/utilities/free_shell.c, 120
run, 14	src/utilities/init_shell.c, 122
shell_t	src/utilities/int_to_str.c, 123
meowsh.h, 48	src/utilities/manage_char_array.c, 123
simple_command.c	src/utilities/my_str_to_word_array.c, 125
add_command, 108	src/utilities/str_alphanum.c, 126
check_input_redirection, 109	src/utilities/strjoin.c, 127
check_output_redirection, 109	start_index
create_command_node, 109	parsing_state_s, 13
handle_no_separator_command, 110	status
src/builtins/alias/alias.c, 68	exec_data_s, 11
src/builtins/alias/handle_alias.c, 69	stock_history
src/builtins/alias/unalias.c, 71	input_s, 11
src/builtins/cd/cd_tilde.c, 71	str_alphanum.c
src/builtins/cd/get_home_user.c, 72	str_isalphanum, 126
src/builtins/cd/my_cd.c, 73	str_input
src/builtins/cd/my_getenv.c, 74	input_s, 12
src/builtins/env/env.c, 75	str_isalphanum
src/builtins/env/setenv.c, 75	meowsh.h, 49
src/builtins/env/unsetenv.c, 78	str_alphanum.c, 126
src/builtins/exec/my_exit.c, 79	strjoin
src/builtins/history/handle_history.c, 80	strjoin.c, 127, 128
src/builtins/which/which.c, 82	utilities.h, 67
src/builtins/which_and_where/where.c, 84	strjoin.c
src/builtins/which_and_where/which.c, 83	strjoin, 127, 128
src/exec/check_signal.c, 85	SUBCMD
src/exec/exec_cmd.c, 86	exec.h, 30
src/exec/get_cmd_path.c, 87	subcommands
src/exec/get_cmd_type.c, 89	commands_s, 8
src/exec/is_built_command.c, 90	- '
src/exec/manage_exec_data.c, 90	tab_builtins_t, 14
src/exec/strjoin.c, 126	builtin_func, 14
src/format/check_exception.c, 92	command, 14
src/format/format.c, 94	local, 15
· · · · · · · · · · · · · · · ·	

```
unalias.c
    interpret_unalias, 71
unsetenv.c
    add_elt, 78
    my_unsetenv, 78
update_quote_state
    check_pipes.c, 114
    parser.h, 63
utilities.h
    del_elt, 65
    dup_array, 65
    free_array, 65
    free_commands, 66
    free_shell, 66
    init_struct, 66
    int_to_str, 66
    my_str_to_word_array, 67
    strjoin, 67
where.c
    interpret_where, 85
    is_a_builtin_command_where, 85
which.c
    interpret_which, 82, 83
    is_a_builtin_command_which, 84
```