

minishell

1

Generated by Doxygen 1.15.0



<b>1 Minishell</b>	<b>1</b>
1.1 Introduction	1
1.2 Features	1
1.3 Usage	1
<b>2 Projet - Minishell</b>	<b>3</b>
2.1 À propos	3
2.2 Compilation du projet	3
2.3 Documentation	3
2.4 Exigences techniques auxquelles nous avons répondu	3
2.4.1 Capacité d'exécuter des commandes simples	3
2.4.2 Capacité d'exécuter un sous-ensemble de plusieurs commandes	4
2.4.3 L'exécution built-in de certaines commandes	4
2.4.4 Le maintien d'un historique	4
2.4.5 Un mode de lancement "batch"	4
2.4.6 La prise en charge d'alias temporaires	5
2.5 Page de manuel linux	5
<b>3 Directory Hierarchy</b>	<b>7</b>
3.1 Directories	7
<b>4 Data Structure Index</b>	<b>9</b>
4.1 Data Structures	9
<b>5 File Index</b>	<b>11</b>
5.1 File List	11
<b>6 Directory Documentation</b>	<b>13</b>
6.1 include Directory Reference	13
6.2 src Directory Reference	13
<b>7 Data Structure Documentation</b>	<b>15</b>
7.1 Alias Struct Reference	15
7.1.1 Detailed Description	15
7.1.2 Field Documentation	15
7.1.2.1 command	15
7.1.2.2 name	15
7.2 BackgroundProcess Struct Reference	16
7.2.1 Detailed Description	16
7.2.2 Field Documentation	16
7.2.2.1 count	16
7.2.2.2 processes	16
7.3 Command Struct Reference	16
7.3.1 Detailed Description	17

7.3.2 Field Documentation	17
7.3.2.1 append_output	17
7.3.2.2 arg_count	17
7.3.2.3 args	17
7.3.2.4 command	17
7.3.2.5 heredoc_delimiter	17
7.3.2.6 input_redirect	17
7.3.2.7 length	17
7.3.2.8 output_redirect	17
7.4 Commands Struct Reference	18
7.4.1 Detailed Description	18
7.4.2 Field Documentation	18
7.4.2.1 command_count	18
7.4.2.2 commands	18
7.4.2.3 operators	18
7.5 EnvVar Struct Reference	18
7.5.1 Field Documentation	19
7.5.1.1 name	19
7.5.1.2 next	19
7.5.1.3 value	19
7.6 Jobs Struct Reference	19
7.6.1 Detailed Description	19
7.6.2 Field Documentation	19
7.6.2.1 command	19
7.6.2.2 pid	19
<b>8 File Documentation</b>	<b>21</b>
8.1 include/aliases.h File Reference	21
8.1.1 Macro Definition Documentation	21
8.1.1.1 MAX_ALIASES	21
8.1.2 Function Documentation	22
8.1.2.1 add_alias()	22
8.1.2.2 free_aliases()	22
8.1.2.3 get_alias_command()	22
8.1.2.4 handle_alias()	22
8.1.2.5 handle_unalias()	23
8.1.2.6 is_alias()	23
8.1.2.7 list_aliases()	23
8.1.2.8 remove_alias()	23
8.2 aliases.h	24
8.3 include/batch.h File Reference	24
8.3.1 Macro Definition Documentation	24

8.3.1.1 MAX_ARGUMENTS . . . . .	24
8.3.2 Function Documentation . . . . .	25
8.3.2.1 handle_arguments() . . . . .	25
8.4 batch.h . . . . .	25
8.5 include/env.h File Reference . . . . .	25
8.5.1 Function Documentation . . . . .	26
8.5.1.1 get_env_var() . . . . .	26
8.5.1.2 print_env_vars() . . . . .	26
8.5.1.3 set_env_var() . . . . .	26
8.5.1.4 unset_env_var() . . . . .	26
8.6 env.h . . . . .	26
8.7 include/executor.h File Reference . . . . .	26
8.7.1 Function Documentation . . . . .	27
8.7.1.1 check_bg_processes() . . . . .	27
8.7.1.2 create_child_process() . . . . .	27
8.7.1.3 execute_command() . . . . .	28
8.7.1.4 execute_commands() . . . . .	28
8.7.1.5 execute_pipes() . . . . .	28
8.7.1.6 handle_heredoc() . . . . .	29
8.7.1.7 setup_redirections() . . . . .	29
8.8 executor.h . . . . .	29
8.9 include/history.h File Reference . . . . .	30
8.9.1 Function Documentation . . . . .	30
8.9.1.1 add_to_history() . . . . .	30
8.9.1.2 handle_history() . . . . .	30
8.10 history.h . . . . .	30
8.11 include/internal_commands.h File Reference . . . . .	31
8.11.1 Function Documentation . . . . .	31
8.11.1.1 free_commands() . . . . .	31
8.11.1.2 free_if_needed() . . . . .	32
8.11.1.3 handle_cd() . . . . .	32
8.11.1.4 handle_echo() . . . . .	32
8.11.1.5 handle_exit() . . . . .	32
8.11.1.6 handle_pwd() . . . . .	33
8.11.2 Variable Documentation . . . . .	33
8.11.2.1 work_dir . . . . .	33
8.12 internal_commands.h . . . . .	33
8.13 include/parser.h File Reference . . . . .	33
8.13.1 Function Documentation . . . . .	34
8.13.1.1 detect_operator() . . . . .	34
8.13.1.2 parse_command() . . . . .	34
8.13.1.3 split_line() . . . . .	35

8.13.1.4 trim_whitespace()	35
8.14 parser.h	35
8.15 include/typedef.h File Reference	36
8.15.1 Macro Definition Documentation	36
8.15.1.1 BLUE	36
8.15.1.2 COLOR_RESET	36
8.15.1.3 GREEN	37
8.15.1.4 MAX_BG_PROCESSES	37
8.15.1.5 MAX_COMMANDS	37
8.15.2 Typedef Documentation	37
8.15.2.1 Alias	37
8.15.2.2 BackgroundProcess	37
8.15.2.3 Command	37
8.15.2.4 Commands	37
8.15.2.5 EnvVar	37
8.15.2.6 Jobs	38
8.16 typedef.h	38
8.17 README.md File Reference	39
8.18 src/aliases.c File Reference	39
8.19 src/batch.c File Reference	39
8.19.1 Detailed Description	39
8.19.2 Function Documentation	40
8.19.2.1 handle_arguments()	40
8.19.3 Variable Documentation	40
8.19.3.1 batch_command	40
8.19.3.2 bg_processes	40
8.19.3.3 minishell_options	40
8.20 src/env.c File Reference	40
8.20.1 Detailed Description	41
8.20.2 Function Documentation	41
8.20.2.1 get_env_var()	41
8.20.2.2 print_env_vars()	41
8.20.2.3 set_env_var()	41
8.20.2.4 unset_env_var()	41
8.20.3 Variable Documentation	42
8.20.3.1 env_vars_head	42
8.20.3.2 env_vars_tail	42
8.21 src/executor.c File Reference	42
8.21.1 Detailed Description	42
8.21.2 Function Documentation	43
8.21.2.1 check_bg_processes()	43
8.21.2.2 create_child_process()	43

8.21.2.3 execute_command()	43
8.21.2.4 execute_commands()	44
8.21.2.5 execute_pipes()	44
8.21.2.6 handle_heredoc()	44
8.21.2.7 setup_redirections()	45
8.21.3 Variable Documentation	45
8.21.3.1 internal_cmds	45
8.21.3.2 internal_cmds_list	45
8.22 src/history.c File Reference	45
8.22.1 Detailed Description	46
8.22.2 Function Documentation	46
8.22.2.1 add_to_history()	46
8.22.2.2 handle_history()	46
8.23 src/internal_commands.c File Reference	46
8.23.1 Detailed Description	47
8.23.2 Function Documentation	47
8.23.2.1 free_commands()	47
8.23.2.2 free_if_needed()	48
8.23.2.3 handle_cd()	48
8.23.2.4 handle_echo()	48
8.23.2.5 handle_exit()	48
8.23.2.6 handle_pwd()	49
8.23.3 Variable Documentation	49
8.23.3.1 work_dir	49
8.24 src/main.c File Reference	49
8.24.1 Detailed Description	49
8.24.2 Function Documentation	50
8.24.2.1 main()	50
8.24.2.2 write_prompt()	50
8.24.3 Variable Documentation	50
8.24.3.1 bg_processes	50
8.24.3.2 parsed_commands	50
8.25 src/parser.c File Reference	50
8.25.1 Detailed Description	51
8.25.2 Function Documentation	51
8.25.2.1 detect_operator()	51
8.25.2.2 parse_command()	51
8.25.2.3 split_line()	52
8.25.2.4 trim_whitespace()	52
8.25.3 Variable Documentation	52
8.25.3.1 OPERATORS	52
8.25.3.2 REDIRECTORS	52





# Chapter 1

## Minishell

### 1.1 Introduction

Welcome to the documentation of the Minishell project for EICNAM 2025.

### 1.2 Features

- [Command](#) parsing and execution
- [Alias](#) management
- Batch mode
- [Command](#) history
- Internal commands (cd, pwd, echo, etc.)

### 1.3 Usage

See the README for usage instructions.



## Chapter 2

# Projet - Minishell

### 2.1 À propos

Ce projet, toujours réalisé dans le cadre de notre module de systèmes avancés, est une version plus légère d'un shell, appelé minishell. Il réponds à la plupart des exigences techniques demandées dans le sujet, celles-ci étant expliquées ci-dessous (lien vers le sujet à l'appui).

### 2.2 Compilation du projet

Le projet se compile avec le fichier Makefile situé dans le même dossier, à l'aide de la commande `make`. Le fichier exécutable "minishell" se trouvera alors dans le sous-dossier `output/`, prêt à l'emploi.

### 2.3 Documentation

- [Documentation générée par doxygen **A REMPLIR QUAND ON AURA LES DONNEES**]()
- [PDF généré par doxygen](#)
- [Sujet du projet](#)
- [Page de manuel linux](#)

### 2.4 Exigences techniques auxquelles nous avons répondu

#### 2.4.1 Capacité d'exécuter des commandes simples

Le minishell réalisé permet l'exécution de commandes simples.

Exemples :

- `ls -a`
- `ps`
- `ls -alF > file.txt`
- `ps aux | grep minishell`

## 2.4.2 Capacité d'exécuter un sous-ensemble de plusieurs commandes

La logique de différents opérateurs a été recréé, permettant l'accès aux :

- Opérateurs de contrôle
  - Le ET logique: `&&`
  - Le OU logique: `||`
- Opérateurs de redirection de flux simples :
  - Le pipe: `|`
  - Les chevrons: `< et >`
  - Les chevrons doubles: `<< et >>`
- Exécutions en arrière-plan
  - Symbole `&`

## 2.4.3 L'exécution built-in de certaines commandes

La logique des commandes suivantes a été réécrite :

- **cd**
- **pwd**
- **exit**
- **echo**

## 2.4.4 Le maintien d'un historique

Un fichier renseignant toutes les commandes rentrées dans le minishell est disponible sous `~/minishell_↵ logs/command_history` à partir du moment où une commande est rentrée dans le terminal. La commande **history** a également été programmée, et permet la visualisation du fichier susmentionné dans le terminal.

## 2.4.5 Un mode de lancement "batch"

Un mode de lancement spécial "batch" a été réalisé, qui permet l'exécution de commandes par argument (qu'elles soient simples ou imbriquées avec des opérateurs).

Pour utiliser le mode batch, la syntaxe est la suivante :

- `./[path_to_minishell_executable] [OPTION] <command>`

Les différentes options possibles sont :

- `-c, --command <command>`: Execute the specified command and exit.
- `--help`: Display help message and exit.

### 2.4.6 La prise en charge d'alias temporaires

Une gestion des alias temporaires, donc disponibles durant une session du minishell, a été prévue (gestion et utilisation des alias dans les commandes lancées).

Les différentes actions possibles sur les alias sont les suivantes :

- La définition d'alias
  - `alias [nom_alias]='[command]'`
- L'affichage de la commande correspondante à un alias
  - `alias [nom_alias]`
- L'affichage de toutes les définitions d'alias
  - `alias`
- La suppression d'alias
  - `unalias [nom_alias]`
- La suppression de tous les alias
  - `unalias -a`

## 2.5 Page de manuel linux

Une page de manuel linux a été créée pour le minishell, et est disponible dans le dossier `doc/` sous le nom `minishell`. Elle peut être consultée en utilisant la commande suivante dans un terminal :

```
man doc/minishell.1
```

Il est possible de l'ajouter au système de manuel linux en copiant le fichier `minishell.1` dans le dossier `/usr/share/man/man1/` (nécessite les droits administrateurs) :

```
sudo cp doc/minishell.1 /usr/share/man/man1/
```

Après cela, la page de manuel pourra être consultée avec la commande :

```
man minishell
```

Si vous ne souhaitez plus avoir accès à la page de manuel, il suffit de supprimer le fichier copié dans `/usr/share/man/man1/` :

```
sudo rm /usr/share/man/man1/minishell.1
```



## Chapter 3

# Directory Hierarchy

### 3.1 Directories

include	13
aliases.h	21
batch.h	24
env.h	25
executor.h	26
history.h	30
internal_commands.h	31
parser.h	33
typedef.h	36
src	13
aliases.c	39
batch.c	39
env.c	40
executor.c	42
history.c	45
internal_commands.c	46
main.c	49
parser.c	50





## Chapter 4

# Data Structure Index

### 4.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">Alias</a>	15
<a href="#">BackgroundProcess</a>	16
<a href="#">Command</a>	16
<a href="#">Commands</a>	18
<a href="#">EnvVar</a>	18
<a href="#">Jobs</a>	19



# Chapter 5

## File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

include/aliases.h	21
include/batch.h	24
include/env.h	25
include/executor.h	26
include/history.h	30
include/internal_commands.h	31
include/parser.h	33
include/typedef.h	36
src/aliases.c	
Implementation of alias handling for the minishell	39
src/batch.c	
Handles batch mode execution for the minishell	39
src/env.c	
Implementation of environment variable management for the minishell	40
src/executor.c	
Implementation of command execution for the minishell	42
src/history.c	
Implementation of command history handling for the minishell	45
src/internal_commands.c	
Implementation of internal commands for the minishell	46
src/main.c	
Main entry point for the minishell application	49
src/parser.c	
Implementation of the command line parser for the minishell	50



## Chapter 6

# Directory Documentation

### 6.1 include Directory Reference

#### Files

- file [aliases.h](#)
- file [batch.h](#)
- file [env.h](#)
- file [executor.h](#)
- file [history.h](#)
- file [internal\\_commands.h](#)
- file [parser.h](#)
- file [typedef.h](#)

### 6.2 src Directory Reference

Directory dependency graph for src:

#### Files

- file [aliases.c](#)  
*Implementation of alias handling for the minishell.*
- file [batch.c](#)  
*Handles batch mode execution for the minishell.*
- file [env.c](#)  
*Implementation of environment variable management for the minishell.*
- file [executor.c](#)  
*Implementation of command execution for the minishell.*
- file [history.c](#)  
*Implementation of command history handling for the minishell.*
- file [internal\\_commands.c](#)  
*Implementation of internal commands for the minishell.*
- file [main.c](#)  
*Main entry point for the minishell application.*
- file [parser.c](#)  
*Implementation of the command line parser for the minishell.*



## Chapter 7

# Data Structure Documentation

### 7.1 Alias Struct Reference

```
#include <typedef.h>
```

#### Data Fields

- char \* [name](#)
- char \* [command](#)

#### 7.1.1 Detailed Description

Structure for alias management

#### 7.1.2 Field Documentation

##### 7.1.2.1 command

```
char* Alias::command
```

##### 7.1.2.2 name

```
char* Alias::name
```

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

- include/[typedef.h](#)

## 7.2 BackgroundProcess Struct Reference

```
#include <typedef.h>
```

Collaboration diagram for BackgroundProcess:

### Data Fields

- int [count](#)
- [Jobs](#) [processes](#) [[MAX\\_BG\\_PROCESSES](#)]

### 7.2.1 Detailed Description

Structure to manage background processes

### 7.2.2 Field Documentation

#### 7.2.2.1 count

```
int BackgroundProcess::count
```

#### 7.2.2.2 processes

```
Jobs BackgroundProcess::processes [MAX\_BG\_PROCESSES]
```

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

- [include/typedef.h](#)

## 7.3 Command Struct Reference

```
#include <typedef.h>
```

### Data Fields

- char \* [command](#)
- size\_t [length](#)
- char \*\* [args](#)
- int [arg\\_count](#)
- char \* [input\\_redirect](#)
- char \* [output\\_redirect](#)
- char \* [heredoc\\_delimiter](#)
- bool [append\\_output](#)



### 7.3.1 Detailed Description

Structure to hold a single command and its details

### 7.3.2 Field Documentation

#### 7.3.2.1 `append_output`

```
bool Command::append_output
```

#### 7.3.2.2 `arg_count`

```
int Command::arg_count
```

#### 7.3.2.3 `args`

```
char** Command::args
```

#### 7.3.2.4 `command`

```
char* Command::command
```

#### 7.3.2.5 `heredoc_delimiter`

```
char* Command::heredoc_delimiter
```

#### 7.3.2.6 `input_redirect`

```
char* Command::input_redirect
```

#### 7.3.2.7 `length`

```
size_t Command::length
```

#### 7.3.2.8 `output_redirect`

```
char* Command::output_redirect
```

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

- [include/typedef.h](#)

## 7.4 Commands Struct Reference

```
#include <typedef.h>
```

Collaboration diagram for Commands:

### Data Fields

- [Command](#) `commands` [[MAX\\_COMMANDS](#)]
- `char *` [operators](#) [[MAX\\_COMMANDS](#)]
- `int` [command\\_count](#)

### 7.4.1 Detailed Description

Structure to hold multiple commands and their operators

### 7.4.2 Field Documentation

#### 7.4.2.1 `command_count`

```
int Commands::command_count
```

#### 7.4.2.2 `commands`

```
Command Commands::commands [MAX\_COMMANDS]
```

#### 7.4.2.3 `operators`

```
char* Commands::operators [MAX\_COMMANDS]
```

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

- [include/typedef.h](#)

## 7.5 EnvVar Struct Reference

```
#include <typedef.h>
```

Collaboration diagram for EnvVar:

### Data Fields

- `char *` [name](#)
- `char *` [value](#)
- `struct` [EnvVar](#) \* [next](#)

## 7.5.1 Field Documentation

### 7.5.1.1 name

```
char* EnvVar::name
```

### 7.5.1.2 next

```
struct EnvVar* EnvVar::next
```

### 7.5.1.3 value

```
char* EnvVar::value
```

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

- include/[typedef.h](#)

## 7.6 Jobs Struct Reference

```
#include <typedef.h>
```

### Data Fields

- [pid\\_t](#) [pid](#)
- char \* [command](#)

## 7.6.1 Detailed Description

Structure to hold background job details

## 7.6.2 Field Documentation

### 7.6.2.1 command

```
char* Jobs::command
```

### 7.6.2.2 pid

```
pid_t Jobs::pid
```

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

- include/[typedef.h](#)



# Chapter 8

## File Documentation

### 8.1 include/aliases.h File Reference

```
#include "typedef.h"  
#include <limits.h>
```

Include dependency graph for aliases.h: This graph shows which files directly or indirectly include this file:

#### Macros

- #define [MAX\\_ALIASES](#) 20

#### Functions

- void [handle\\_alias](#) (char \*\*args)  
*Handles the alias command and its various functionalities.*
- void [handle\\_unalias](#) (char \*\*args)  
*Handles the unalias command and its various functionalities.*
- void [add\\_alias](#) ([Alias](#) new\_alias)  
*Adds a new alias to the alias list.*
- void [remove\\_alias](#) (const char \*name)  
*Removes an alias from the alias list from its name.*
- int [is\\_alias](#) (const char \*name)  
*Checks if a given name corresponds to an existing alias.*
- void [list\\_aliases](#) ()  
*Lists all currently defined aliases. Used when the alias command is called without arguments.*
- void [free\\_aliases](#) ()  
*Frees all allocated memory for aliases and resets the alias count.*
- char \* [get\\_alias\\_command](#) (const char \*name)  
*Retrieves the command associated with a given alias name.*

#### 8.1.1 Macro Definition Documentation

##### 8.1.1.1 MAX\_ALIASES

```
#define MAX_ALIASES 20
```

## 8.1.2 Function Documentation

### 8.1.2.1 add\_alias()

```
void add_alias (  
    Alias new_alias)
```

Adds a new alias to the alias list.

#### Parameters

<i>new_alias</i>	The <a href="#">Alias</a> struct containing the name and command of the new alias.
------------------	--

### 8.1.2.2 free\_aliases()

```
void free_aliases ()
```

Frees all allocated memory for aliases and resets the alias count.

### 8.1.2.3 get\_alias\_command()

```
char * get_alias_command (  
    const char * name)
```

Retrieves the command associated with a given alias name.

#### Parameters

<i>name</i>	The name of the alias.
-------------	------------------------

#### Returns

The command string if the alias exists, NULL otherwise.

### 8.1.2.4 handle\_alias()

```
void handle_alias (  
    char ** args)
```

Handles the alias command and its various functionalities.

#### Parameters

<i>args</i>	The different arguments passed to the alias command.
-------------	--

#### 8.1.2.5 handle\_unalias()

```
void handle_unalias (
    char ** args)
```

Handles the unalias command and its various functionalities.

##### Parameters

<i>args</i>	The different arguments passed to the unalias command (an alias name or -a).
-------------	--

#### 8.1.2.6 is\_alias()

```
int is_alias (
    const char * name)
```

Checks if a given name corresponds to an existing alias.

##### Parameters

<i>name</i>	The name to check.
-------------	--------------------

##### Returns

EXIT\_SUCCESS if the name is an alias, EXIT\_FAILURE otherwise.

#### 8.1.2.7 list\_aliases()

```
void list_aliases ()
```

Lists all currently defined aliases. Used when the alias command is called without arguments.

#### 8.1.2.8 remove\_alias()

```
void remove_alias (
    const char * name)
```

Removes an alias from the alias list from its name.

##### Parameters

<i>name</i>	The name of the alias to remove.
-------------	----------------------------------

## 8.2 aliases.h

[Go to the documentation of this file.](#)

```

00001 #ifndef ALIASES_H
00002 #define ALIASES_H
00003
00004 #include "typedef.h"
00005 #include <limits.h>
00006
00007 #define MAX_ALIASES 20
00008
00013 void handle_alias(char **args);
00014
00019 void handle_unalias(char **args);
00020
00025 void add_alias(Alias new_alias);
00026
00031 void remove_alias(const char *name);
00032
00038 int is_alias(const char *name);
00039
00044 void list_aliases();
00045
00049 void free_aliases();
00050
00056 char* get_alias_command(const char *name);
00057
00058 #endif // ALIASES_H

```

## 8.3 include/batch.h File Reference

```

#include "typedef.h"
#include "internal_commands.h"
#include "parser.h"
#include "executor.h"

```

Include dependency graph for batch.h: This graph shows which files directly or indirectly include this file:

### Macros

- `#define MAX_ARGUMENTS 3`

### Functions

- `int handle_arguments (int argc, const char *argv[ ])`  
*Handles command-line arguments for batch mode. Possible options are:*

### 8.3.1 Macro Definition Documentation

#### 8.3.1.1 MAX\_ARGUMENTS

```
#define MAX_ARGUMENTS 3
```



## 8.3.2 Function Documentation

### 8.3.2.1 handle\_arguments()

```
int handle_arguments (
    int argc,
    const char * argv[])
```

Handles command-line arguments for batch mode. Possible options are:

- `-c, --command <command>`: Execute the specified command and exit.
- `--help`: Display help message and exit.

#### Parameters

<i>argc</i>	Argument count.
<i>argv</i>	Argument vector.

#### Returns

Returns EXIT\_SUCCESS or EXIT\_FAILURE.

## 8.4 batch.h

[Go to the documentation of this file.](#)

```
00001 #ifndef BATCH_H
00002 #define BATCH_H
00003
00004 #include "typedef.h"
00005 #include "internal_commands.h"
00006 #include "parser.h"
00007 #include "executor.h"
00008
00009 #define MAX_ARGUMENTS 3
00010
00020 int handle_arguments(int argc, const char *argv[]);
00021
00022 #endif // BATCH_H
```

## 8.5 include/env.h File Reference

```
#include "typedef.h"
```

Include dependency graph for env.h: This graph shows which files directly or indirectly include this file:

#### Functions

- void [set\\_env\\_var](#) (const char \*name, const char \*value)
- void [unset\\_env\\_var](#) (const char \*name)
- char \* [get\\_env\\_var](#) (const char \*name)
- void [print\\_env\\_vars](#) ()

## 8.5.1 Function Documentation

### 8.5.1.1 `get_env_var()`

```
char * get_env_var (
    const char * name)
```

### 8.5.1.2 `print_env_vars()`

```
void print_env_vars ()
```

### 8.5.1.3 `set_env_var()`

```
void set_env_var (
    const char * name,
    const char * value)
```

### 8.5.1.4 `unset_env_var()`

```
void unset_env_var (
    const char * name)
```

## 8.6 `env.h`

[Go to the documentation of this file.](#)

```
00001 #ifndef ENV_H
00002 #define ENV_H
00003
00004 #include "typedef.h"
00005
00006 void set_env_var(const char* name, const char* value);
00007 void unset_env_var(const char* name);
00008
00009 char* get_env_var(const char* name);
00010 void print_env_vars();
00011
00012 #endif // ENV_H
```

## 8.7 `include/executor.h` File Reference

```
#include "typedef.h"
#include "internal_commands.h"
#include "aliases.h"
#include "history.h"
```

Include dependency graph for `executor.h`: This graph shows which files directly or indirectly include this file:

## Functions

- int `execute_command` (`Command` \*command, bool is\_background)  
*Executes a single command, checking for internal commands first.*
- void `execute_commands` (`Commands` \*commands, `BackgroundProcess` \*bg\_processes)  
*Executes a series of commands with their associated operators.*
- int `create_child_process` (`Command` \*command, bool is\_background)  
*Creates a child process to execute a command.*
- void `check_bg_processes` (`BackgroundProcess` \*bg\_processes)  
*Checks the status of background processes and cleans up finished ones.*
- int `execute_pipes` (`Commands` \*commands, int start\_index, int end\_index)  
*Executes a series of piped commands.*
- int `setup_redirections` (`Command` \*command)  
*Sets up input and output redirections for a command.*
- int `handle_heredoc` (`Command` \*command)  
*Handles here-document (heredoc) input for a command.*

## 8.7.1 Function Documentation

### 8.7.1.1 `check_bg_processes()`

```
void check_bg_processes (  
    BackgroundProcess * bg_processes)
```

Checks the status of background processes and cleans up finished ones.

#### Parameters

<code>bg_processes</code>	Pointer to the <code>BackgroundProcess</code> struct managing background jobs.
---------------------------	--

### 8.7.1.2 `create_child_process()`

```
int create_child_process (  
    Command * command,  
    bool is_background)
```

Creates a child process to execute a command.

#### Parameters

<code>command</code>	Pointer to the <code>Command</code> struct to execute.
<code>is_background</code>	Boolean indicating if the command should run in the background.

#### Returns

Exit status of the command execution.

### 8.7.1.3 execute\_command()

```
int execute_command (
    Command * command,
    bool is_background)
```

Executes a single command, checking for internal commands first.

#### Parameters

<i>command</i>	Pointer to the <a href="#">Command</a> struct to execute.
<i>is_background</i>	Boolean indicating if the command should run in the background.

#### Returns

Exit status of the command execution.

### 8.7.1.4 execute\_commands()

```
void execute_commands (
    Commands * commands,
    BackgroundProcess * bg_processes)
```

Executes a series of commands with their associated operators.

#### Parameters

<i>commands</i>	Pointer to the <a href="#">Commands</a> struct containing commands and operators.
-----------------	---

### 8.7.1.5 execute\_pipes()

```
int execute_pipes (
    Commands * commands,
    int start_index,
    int end_index)
```

Executes a series of piped commands.

#### Parameters

<i>commands</i>	Pointer to the <a href="#">Commands</a> struct containing commands and operators.
<i>start_index</i>	Index of the first command in the pipe sequence.
<i>end_index</i>	Index of the last command in the pipe sequence.

#### Returns

Exit status of the last command in the pipe sequence.

### 8.7.1.6 handle\_heredoc()

```
int handle_heredoc (
    Command * command)
```

Handles here-document (heredoc) input for a command.

#### Parameters

<i>command</i>	Pointer to the <a href="#">Command</a> struct containing the heredoc delimiter.
----------------	---

#### Returns

0 on success, -1 on failure.

### 8.7.1.7 setup\_redirections()

```
int setup_redirections (
    Command * command)
```

Sets up input and output redirections for a command.

#### Parameters

<i>command</i>	Pointer to the <a href="#">Command</a> struct to set up redirections for.
----------------	---

#### Returns

0 on success, -1 on failure.

## 8.8 executor.h

[Go to the documentation of this file.](#)

```
00001 #ifndef EXECUTOR_H
00002 #define EXECUTOR_H
00003
00004 #include "typedef.h"
00005 #include "internal_commands.h"
00006 #include "aliases.h"
00007 #include "history.h"
00008
00015 int execute_command(Command* command, bool is_background);
00016
00021 void execute_commands(Commands* commands, BackgroundProcess* bg_processes);
00022
00029 int create_child_process(Command* command, bool is_background);
00030
00035 void check_bg_processes(BackgroundProcess* bg_processes);
00036
00044 int execute_pipes(Commands* commands, int start_index, int end_index);
00045
00051 int setup_redirections(Command* command);
00052
00058 int handle_heredoc(Command* command);
00059
00060 #endif // EXECUTOR_H
```

## 8.9 include/history.h File Reference

```
#include "typedef.h"
```

Include dependency graph for history.h: This graph shows which files directly or indirectly include this file:

### Functions

- void [handle\\_history](#) (char \*\*args)  
*Built-in history command handler. Handles displaying the command history to the user.*
- void [add\\_to\\_history](#) (char \*command\_line)  
*Handles writing inputted commands in history file (~/.minishell\_logs/command\_history).*

### 8.9.1 Function Documentation

#### 8.9.1.1 add\_to\_history()

```
void add_to_history (
    char * command_line)
```

Handles writing inputted commands in history file (~/.minishell\_logs/command\_history).

#### Parameters

<i>command_line</i>	The entire command line that needs to be added in the history file.
---------------------	---

#### 8.9.1.2 handle\_history()

```
void handle_history (
    char ** args)
```

Built-in history command handler. Handles displaying the command history to the user.

#### Parameters

<i>args</i>	voided
-------------	--------

## 8.10 history.h

[Go to the documentation of this file.](#)

```
00001 #ifndef HISTORY_H
00002 #define HISTORY_H
00003
00004 #include "typedef.h"
00005
00011 void handle_history(char** args);
00012
00017 void add_to_history(char* command_line);
00018
00019 #endif // HISTORY_H
```

## 8.11 include/internal\_commands.h File Reference

```
#include "typedef.h"
```

```
#include "aliases.h"
```

Include dependency graph for internal\_commands.h: This graph shows which files directly or indirectly include this file:

### Functions

- void [handle\\_cd](#) (char \*\*args)  
*Built-in cd command handler.*
- void [handle\\_pwd](#) (char \*\*args)  
*Built-in pwd command handler.*
- void [handle\\_echo](#) (char \*\*args)  
*Built-in echo command handler.*
- void [handle\\_exit](#) ([Commands](#) \*commands)  
*Built-in exit command handler. Exits the shell after freeing necessary resources.*
- void [free\\_if\\_needed](#) (void \*\_free)  
*Frees memory if the pointer is not NULL.*
- void [free\\_commands](#) ([Commands](#) \*commands)  
*Frees all allocated memory in the [Commands](#) struct.*

### Variables

- char \* [work\\_dir](#)

### 8.11.1 Function Documentation

#### 8.11.1.1 [free\\_commands\(\)](#)

```
void free_commands (  
    Commands * commands)
```

Frees all allocated memory in the [Commands](#) struct.

#### Parameters

<i>commands</i>	Pointer to the <a href="#">Commands</a> struct to free.
-----------------	---

#### 8.11.1.2 free\_if\_needed()

```
void free_if_needed (
    void * to_free)
```

Frees memory if the pointer is not NULL.

##### Parameters

<i>to_free</i>	Pointer to the memory to free.
----------------	--------------------------------

#### 8.11.1.3 handle\_cd()

```
void handle_cd (
    char ** args)
```

Built-in cd command handler.

##### Parameters

<i>args</i>	<a href="#">Command</a> arguments, handles correctly changing directories, and specific cases like "cd ~".
-------------	--

#### 8.11.1.4 handle\_echo()

```
void handle_echo (
    char ** args)
```

Built-in echo command handler.

##### Parameters

<i>args</i>	<a href="#">Command</a> arguments to echo back to the user.
-------------	---

#### 8.11.1.5 handle\_exit()

```
void handle_exit (
    Commands * commands)
```

Built-in exit command handler. Exits the shell after freeing necessary resources.

##### Parameters

<i>commands</i>	Pointer to the <a href="#">Commands</a> struct containing commands and operators.
-----------------	---



### 8.11.1.6 handle\_pwd()

```
void handle_pwd (
    char ** args)
```

Built-in pwd command handler.

#### Parameters

<i>args</i>	voided
-------------	--------

## 8.11.2 Variable Documentation

### 8.11.2.1 work\_dir

```
char* work_dir [extern]
```

The path to the current workdir

## 8.12 internal\_commands.h

[Go to the documentation of this file.](#)

```
00001 #ifndef INTERNAL_COMMANDS_H
00002 #define INTERNAL_COMMANDS_H
00003
00004 #include "typedef.h"
00005 #include "aliases.h"
00006
00007 extern char *work_dir;
00008
00013 void handle_cd(char** args);
00014
00019 void handle_pwd(char** args);
00020
00025 void handle_echo(char** args);
00026
00032 void handle_exit(Commands* commands);
00033
00038 void free_if_needed(void* to_free);
00039
00044 void free_commands(Commands* commands);
00045
00046 #endif // INTERNAL_COMMANDS_H
```

## 8.13 include/parser.h File Reference

```
#include "typedef.h"
#include "internal_commands.h"
```

Include dependency graph for parser.h: This graph shows which files directly or indirectly include this file:

## Functions

- int [detect\\_operator](#) (const char \*p, const char \*\*op\_found)  
*Detects if the current position in the input line matches any known operator.*
- char \* [trim\\_whitespace](#) (const char \*line)  
*Trims leading and trailing whitespace from a given line.*
- void [split\\_line](#) (const char \*line, [Commands](#) \*commands)  
*Splits the input line into commands and operators, populating the [Commands](#) struct.*
- void [parse\\_command](#) ([Command](#) \*command)  
*Parses a command string into its arguments and populates the [Command](#) struct.*

## 8.13.1 Function Documentation

### 8.13.1.1 detect\_operator()

```
int detect_operator (
    const char * p,
    const char ** op_found)
```

Detects if the current position in the input line matches any known operator.

#### Parameters

<i>p</i>	Pointer to the current position in the input line.
<i>op_found</i>	Pointer to store the found operator string.

#### Returns

Length of the detected operator, or 0 if none found.

### 8.13.1.2 parse\_command()

```
void parse_command (
    Command * command)
```

Parses a command string into its arguments and populates the [Command](#) struct.

#### Parameters

<i>command</i>	Pointer to the <a href="#">Command</a> struct to populate.
----------------	--

### 8.13.1.3 split\_line()

```
void split_line (
    const char * line,
    Commands * commands)
```

Splits the input line into commands and operators, populating the [Commands](#) struct.

#### Parameters

<i>line</i>	Pointer to the input line.
<i>commands</i>	Pointer to the <a href="#">Commands</a> struct to populate.

### 8.13.1.4 trim\_whitespace()

```
char * trim_whitespace (
    const char * line)
```

Trims leading and trailing whitespace from a given line.

#### Parameters

<i>line</i>	Pointer to the input line.
-------------	----------------------------

#### Returns

Newly allocated string with trimmed whitespace.

## 8.14 parser.h

[Go to the documentation of this file.](#)

```
00001 #ifndef PARSER_H
00002 #define PARSER_H
00003
00004 #include "typedef.h"
00005 #include "internal_commands.h"
00006
00013 int detect_operator(const char* p, const char** op_found);
00014
00020 char* trim_whitespace(const char* line);
00021
00027 void split_line(const char* line, Commands *commands);
00028
00033 void parse_command(Command* command);
00034
00035 #endif // PARSER_H
```

## 8.15 include/typedef.h File Reference

```
#include <string.h>
#include <stdbool.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/types.h>
```

Include dependency graph for typedef.h: This graph shows which files directly or indirectly include this file:

### Data Structures

- struct [Command](#)
- struct [Commands](#)
- struct [Jobs](#)
- struct [BackgroundProcess](#)
- struct [Alias](#)
- struct [EnvVar](#)

### Macros

- #define [MAX\\_COMMANDS](#) 3
- #define [MAX\\_BG\\_PROCESSES](#) 10
- #define [GREEN](#) "\033[1;32m"
- #define [BLUE](#) "\033[1;34m"
- #define [COLOR\\_RESET](#) "\033[0m"

### Typedefs

- typedef struct Command [Command](#)
- typedef struct Commands [Commands](#)
- typedef struct Jobs [Jobs](#)
- typedef struct BackgroundProcess [BackgroundProcess](#)
- typedef struct Alias [Alias](#)
- typedef struct EnvVar [EnvVar](#)

## 8.15.1 Macro Definition Documentation

### 8.15.1.1 BLUE

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

### 8.15.1.2 COLOR\_RESET

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

### 8.15.1.3 GREEN

```
#define GREEN "\033[1;32m"
```

### 8.15.1.4 MAX\_BG\_PROCESSES

```
#define MAX_BG_PROCESSES 10
```

### 8.15.1.5 MAX\_COMMANDS

```
#define MAX_COMMANDS 3
```

## 8.15.2 Typedef Documentation

### 8.15.2.1 Alias

```
typedef struct Alias Alias
```

Structure for alias management

### 8.15.2.2 BackgroundProcess

```
typedef struct BackgroundProcess BackgroundProcess
```

Structure to manage background processes

### 8.15.2.3 Command

```
typedef struct Command Command
```

Structure to hold a single command and its details

### 8.15.2.4 Commands

```
typedef struct Commands Commands
```

Structure to hold multiple commands and their operators

### 8.15.2.5 EnvVar

```
typedef struct EnvVar EnvVar
```

### 8.15.2.6 Jobs

```
typedef struct Jobs Jobs
```

Structure to hold background job details

## 8.16 typedef.h

[Go to the documentation of this file.](#)

```
00001 #ifndef TYPEDEF_H
00002 #define TYPEDEF_H
00003
00004
00021 #include <string.h>
00022 #include <stdbool.h>
00023 #include <unistd.h>
00024 #include <stdio.h>
00025 #include <stdlib.h>
00026 #include <sys/wait.h>
00027 #include <fcntl.h>
00028 #include <errno.h>
00029 #include <sys/stat.h>
00030 #include <sys/types.h>
00031
00032 #define MAX_COMMANDS      3
00033 #define MAX_BG_PROCESSES  10
00034
00035 #define GREEN             "\033[1;32m"
00036 #define BLUE              "\033[1;34m"
00037 #define COLOR_RESET      "\033[0m"
00038
00040 typedef struct Command {
00041     char*    command;           // Original command string
00042     size_t   length;           // Length of the command string
00043     char**   args;             // Argument list
00044     int      arg_count;        // Number of arguments related to the command
00045     char*    input_redirect;   // file for <
00046     char*    output_redirect;  // file for > or »
00047     char*    heredoc_delimiter; // delimiter for «
00048     bool     append_output;    // true for », false for >
00049 } Command;
00050
00052 typedef struct Commands {
00053     Command commands[MAX_COMMANDS]; // Array of parsed commands
00054     char*    operators[MAX_COMMANDS]; // Operators between commands
00055     int      command_count;          // Number of commands parsed
00056 } Commands;
00057
00059 typedef struct Jobs {
00060     pid_t   pid;           // Process ID
00061     char*    command;      // Command string
00062 } Jobs;
00063
00065 typedef struct BackgroundProcess {
00066     int      count;        // Number of background processes
00067     Jobs     processes[MAX_BG_PROCESSES]; // Array of background processes
00068 } BackgroundProcess;
00069
00071 typedef struct Alias {
00072     char*    name;         // Alias name
00073     char*    command;      // Command string
00074 } Alias;
00075
00076 typedef struct EnvVar {
00077     char*    name;         // Environment variable name
00078     char*    value;        // Environment variable value
00079     struct EnvVar* next;    // Pointer to the next variable in the list
00080 } EnvVar;
00081
00082 #endif // TYPEDEF_H
```

## 8.17 README.md File Reference

## 8.18 src/aliases.c File Reference

Implementation of alias handling for the minishell.

```
#include "../include/aliases.h"  
Include dependency graph for aliases.c:
```

## 8.19 src/batch.c File Reference

Handles batch mode execution for the minishell.

```
#include "../include/batch.h"  
Include dependency graph for batch.c:
```

### Functions

- int [handle\\_arguments](#) (int argc, const char \*argv[])  
*Handles command-line arguments for batch mode. Possible options are:*

### Variables

- const char \* [minishell\\_options](#) [] = {"-c", "--command", "--help", NULL}
- [Commands](#) [batch\\_command](#)
- [BackgroundProcess](#) [bg\\_processes](#)

### 8.19.1 Detailed Description

Handles batch mode execution for the minishell.

#### Author

BRENNER Quentin, NEAGELY Jeannot

#### Date

2025-2026

## 8.19.2 Function Documentation

### 8.19.2.1 `handle_arguments()`

```
int handle_arguments (
    int argc,
    const char * argv[])
```

Handles command-line arguments for batch mode. Possible options are:

- `-c, --command <command>`: Execute the specified command and exit.
- `--help`: Display help message and exit.

#### Parameters

<code>argc</code>	Argument count.
<code>argv</code>	Argument vector.

#### Returns

Returns `EXIT_SUCCESS` or `EXIT_FAILURE`.

## 8.19.3 Variable Documentation

### 8.19.3.1 `batch_command`

`Commands` `batch_command`

The command to be run

### 8.19.3.2 `bg_processes`

`BackgroundProcess` `bg_processes` [extern]

### 8.19.3.3 `minishell_options`

```
const char* minishell_options[] = {"-c", "--command", "--help", NULL}
```

List of possible options to use for batch mode

## 8.20 `src/env.c` File Reference

Implementation of environment variable management for the minishell.

```
#include "../include/env.h"
Include dependency graph for env.c:
```



## Functions

- void [set\\_env\\_var](#) (const char \*name, const char \*value)
- void [unset\\_env\\_var](#) (const char \*name)
- char \* [get\\_env\\_var](#) (const char \*name)
- void [print\\_env\\_vars](#) ()

## Variables

- [EnvVar](#) \* [env\\_vars\\_head](#) = NULL
- [EnvVar](#) \* [env\\_vars\\_tail](#) = NULL

### 8.20.1 Detailed Description

Implementation of environment variable management for the minishell.

#### Author

BRENNER Quentin, NEAGELY Jeannot

#### Date

2025-2026

### 8.20.2 Function Documentation

#### 8.20.2.1 [get\\_env\\_var\(\)](#)

```
char * get\_env\_var (  
    const char * name)
```

#### 8.20.2.2 [print\\_env\\_vars\(\)](#)

```
void print\_env\_vars ()
```

#### 8.20.2.3 [set\\_env\\_var\(\)](#)

```
void set\_env\_var (  
    const char * name,  
    const char * value)
```

#### 8.20.2.4 [unset\\_env\\_var\(\)](#)

```
void unset\_env\_var (  
    const char * name)
```

## 8.20.3 Variable Documentation

### 8.20.3.1 env\_vars\_head

```
EnvVar* env_vars_head = NULL
```

### 8.20.3.2 env\_vars\_tail

```
EnvVar* env_vars_tail = NULL
```

## 8.21 src/executor.c File Reference

Implementation of command execution for the minishell.

```
#include "../include/executor.h"
```

Include dependency graph for executor.c:

### Functions

- int [create\\_child\\_process](#) ([Command](#) \*command, bool is\_background)  
*Creates a child process to execute a command.*
- void [execute\\_commands](#) ([Commands](#) \*commands, [BackgroundProcess](#) \*bg\_processes)  
*Executes a series of commands with their associated operators.*
- int [execute\\_command](#) ([Command](#) \*command, bool is\_background)  
*Executes a single command, checking for internal commands first.*
- void [check\\_bg\\_processes](#) ([BackgroundProcess](#) \*bg\_processes)  
*Checks the status of background processes and cleans up finished ones.*
- int [execute\\_pipes](#) ([Commands](#) \*commands, int start\_index, int end\_index)  
*Executes a series of piped commands.*
- int [setup\\_redirections](#) ([Command](#) \*command)  
*Sets up input and output redirections for a command.*
- int [handle\\_heredoc](#) ([Command](#) \*command)  
*Handles here-document (heredoc) input for a command.*

### Variables

- const char \* [internal\\_cmds\\_list](#) [] = {"cd", "pwd", "echo", "history", "alias", "unalias", NULL}
- void(\* [internal\\_cmds](#) [])(char \*\*) = {&[handle\\_cd](#), &[handle\\_pwd](#), &[handle\\_echo](#), &[handle\\_history](#), &[handle\\_alias](#), &[handle\\_unalias](#), NULL}

### 8.21.1 Detailed Description

Implementation of command execution for the minishell.

#### Author

BRENNER Quentin, NEAGELY Jeannot

#### Date

2025-2026

## 8.21.2 Function Documentation

### 8.21.2.1 check\_bg\_processes()

```
void check_bg_processes (  
    BackgroundProcess * bg_processes)
```

Checks the status of background processes and cleans up finished ones.

#### Parameters

<i>bg_processes</i>	Pointer to the <a href="#">BackgroundProcess</a> struct managing background jobs.
---------------------	---

### 8.21.2.2 create\_child\_process()

```
int create_child_process (  
    Command * command,  
    bool is_background)
```

Creates a child process to execute a command.

#### Parameters

<i>command</i>	Pointer to the <a href="#">Command</a> struct to execute.
<i>is_background</i>	Boolean indicating if the command should run in the background.

#### Returns

Exit status of the command execution.

### 8.21.2.3 execute\_command()

```
int execute_command (  
    Command * command,  
    bool is_background)
```

Executes a single command, checking for internal commands first.

#### Parameters

<i>command</i>	Pointer to the <a href="#">Command</a> struct to execute.
<i>is_background</i>	Boolean indicating if the command should run in the background.

#### Returns

Exit status of the command execution.

#### 8.21.2.4 execute\_commands()

```
void execute_commands (
    Commands * commands,
    BackgroundProcess * bg_processes)
```

Executes a series of commands with their associated operators.

##### Parameters

<i>commands</i>	Pointer to the <a href="#">Commands</a> struct containing commands and operators.
-----------------	---

#### 8.21.2.5 execute\_pipes()

```
int execute_pipes (
    Commands * commands,
    int start_index,
    int end_index)
```

Executes a series of piped commands.

##### Parameters

<i>commands</i>	Pointer to the <a href="#">Commands</a> struct containing commands and operators.
<i>start_index</i>	Index of the first command in the pipe sequence.
<i>end_index</i>	Index of the last command in the pipe sequence.

##### Returns

Exit status of the last command in the pipe sequence.

#### 8.21.2.6 handle\_heredoc()

```
int handle_heredoc (
    Command * command)
```

Handles here-document (heredoc) input for a command.

##### Parameters

<i>command</i>	Pointer to the <a href="#">Command</a> struct containing the heredoc delimiter.
----------------	---

##### Returns

0 on success, -1 on failure.

### 8.21.2.7 setup\_redirections()

```
int setup_redirections (
    Command * command)
```

Sets up input and output redirections for a command.

#### Parameters

<i>command</i>	Pointer to the <a href="#">Command</a> struct to set up redirections for.
----------------	---

#### Returns

0 on success, -1 on failure.

## 8.21.3 Variable Documentation

### 8.21.3.1 internal\_cmds

```
void(* internal_cmds[])(char **) (
    char ** ) = {&handle_cd, &handle_pwd, &handle_echo, &handle_history, &handle_alias,
&handle_unalias, NULL}
```

List of pointers to internal command functions

### 8.21.3.2 internal\_cmds\_list

```
const char* internal_cmds_list[] = {"cd", "pwd", "echo", "history", "alias", "unalias", NULL}
```

List of internal commands

## 8.22 src/history.c File Reference

Implementation of command history handling for the minishell.

```
#include "../include/history.h"
Include dependency graph for history.c:
```

#### Functions

- void [handle\\_history](#) (char \*\*args)  
*Built-in history command handler. Handles displaying the command history to the user.*
- void [add\\_to\\_history](#) (char \*command\_line)  
*Handles writing inputted commands in history file (~/.minishell\_logs/command\_history).*

### 8.22.1 Detailed Description

Implementation of command history handling for the minishell.

#### Author

BRENNER Quentin, NEAGELY Jeannot

#### Date

2025-2026

### 8.22.2 Function Documentation

#### 8.22.2.1 `add_to_history()`

```
void add_to_history (
    char * command_line)
```

Handles writing inputted commands in history file (`~/minishell_logs/command_history`).

#### Parameters

<i>command_line</i>	The entire command line that needs to be added in the history file.
---------------------	---

#### 8.22.2.2 `handle_history()`

```
void handle_history (
    char ** args)
```

Built-in history command handler. Handles displaying the command history to the user.

#### Parameters

<i>args</i>	voided
-------------	--------

## 8.23 `src/internal_commands.c` File Reference

Implementation of internal commands for the minishell.

```
#include "../include/internal_commands.h"
Include dependency graph for internal_commands.c:
```

## Functions

- void `handle_exit` (`Commands` \*commands)  
*Built-in exit command handler. Exits the shell after freeing necessary resources.*
- void `handle_pwd` (char \*\*args)  
*Built-in pwd command handler.*
- void `handle_cd` (char \*\*args)  
*Built-in cd command handler.*
- void `handle_echo` (char \*\*args)  
*Built-in echo command handler.*
- void `free_if_needed` (void \*\_to\_free)  
*Frees memory if the pointer is not NULL.*
- void `free_commands` (`Commands` \*commands)  
*Frees all allocated memory in the `Commands` struct.*

## Variables

- char \* `work_dir`

### 8.23.1 Detailed Description

Implementation of internal commands for the minishell.

#### Author

BRENNER Quentin, NEAGELY Jeannot

#### Date

2025-2026

### 8.23.2 Function Documentation

#### 8.23.2.1 `free_commands()`

```
void free_commands (  
    Commands * commands)
```

Frees all allocated memory in the `Commands` struct.

#### Parameters

<code>commands</code>	Pointer to the <code>Commands</code> struct to free.
-----------------------	--

### 8.23.2.2 free\_if\_needed()

```
void free_if_needed (
    void * to_free)
```

Frees memory if the pointer is not NULL.

#### Parameters

<i>to_free</i>	Pointer to the memory to free.
----------------	--------------------------------

### 8.23.2.3 handle\_cd()

```
void handle_cd (
    char ** args)
```

Built-in cd command handler.

#### Parameters

<i>args</i>	<a href="#">Command</a> arguments, handles correctly changing directories, and specific cases like "cd ~".
-------------	--

### 8.23.2.4 handle\_echo()

```
void handle_echo (
    char ** args)
```

Built-in echo command handler.

#### Parameters

<i>args</i>	<a href="#">Command</a> arguments to echo back to the user.
-------------	---

### 8.23.2.5 handle\_exit()

```
void handle_exit (
    Commands * commands)
```

Built-in exit command handler. Exits the shell after freeing necessary resources.

#### Parameters

<i>commands</i>	Pointer to the <a href="#">Commands</a> struct containing commands and operators.
-----------------	---



### 8.23.2.6 handle\_pwd()

```
void handle_pwd (  
    char ** args)
```

Built-in pwd command handler.

#### Parameters

<i>args</i>	voided
-------------	--------

## 8.23.3 Variable Documentation

### 8.23.3.1 work\_dir

```
char* work_dir
```

The path to the current workdir

## 8.24 src/main.c File Reference

Main entry point for the minishell application.

```
#include "../include/typedef.h"  
#include "../include/parser.h"  
#include "../include/internal_commands.h"  
#include "../include/executor.h"  
#include "../include/batch.h"
```

Include dependency graph for main.c:

#### Functions

- void [write\\_prompt](#) (bool is\_compact)
- int [main](#) (int argc, const char \*argv[])

#### Variables

- [Commands parsed\\_commands](#)
- [BackgroundProcess bg\\_processes](#) = {.count = 0}

### 8.24.1 Detailed Description

Main entry point for the minishell application.

#### Author

BRENNER Quentin, NEAGELY Jeannot

#### Date

2025-2026

## 8.24.2 Function Documentation

### 8.24.2.1 main()

```
int main (
    int argc,
    const char * argv[])
```

### 8.24.2.2 write\_prompt()

```
void write_prompt (
    bool is_compact)
```

## 8.24.3 Variable Documentation

### 8.24.3.1 bg\_processes

```
BackgroundProcess bg_processes = {.count = 0}
```

### 8.24.3.2 parsed\_commands

```
Commands parsed_commands
```

## 8.25 src/parser.c File Reference

Implementation of the command line parser for the minishell.

```
#include "../include/parser.h"
Include dependency graph for parser.c:
```

### Functions

- int [detect\\_operator](#) (const char \*p, const char \*\*op\_found)  
*Detects if the current position in the input line matches any known operator.*
- char \* [trim\\_whitespace](#) (const char \*line)  
*Trims leading and trailing whitespace from a given line.*
- void [split\\_line](#) (const char \*line, [Commands](#) \*commands)  
*Splits the input line into commands and operators, populating the [Commands](#) struct.*
- void [parse\\_command](#) ([Command](#) \*command)  
*Parses a command string into its arguments and populates the [Command](#) struct.*

### Variables

- const char \* [OPERATORS](#) [] = {"&&", "|", "&", "|", ":", NULL}
- const char \* [REDIRECTORS](#) [] = {"<<", ">>", "<", ">", NULL}

## 8.25.1 Detailed Description

Implementation of the command line parser for the minishell.

### Author

BRENNER Quentin, NEAGELY Jeannot

### Date

2025-2026

## 8.25.2 Function Documentation

### 8.25.2.1 detect\_operator()

```
int detect_operator (
    const char * p,
    const char ** op_found)
```

Detects if the current position in the input line matches any known operator.

#### Parameters

<i>p</i>	Pointer to the current position in the input line.
<i>op_found</i>	Pointer to store the found operator string.

#### Returns

Length of the detected operator, or 0 if none found.

### 8.25.2.2 parse\_command()

```
void parse_command (
    Command * command)
```

Parses a command string into its arguments and populates the [Command](#) struct.

#### Parameters

<i>command</i>	Pointer to the <a href="#">Command</a> struct to populate.
----------------	--

### 8.25.2.3 split\_line()

```
void split_line (
    const char * line,
    Commands * commands)
```

Splits the input line into commands and operators, populating the [Commands](#) struct.

#### Parameters

<i>line</i>	Pointer to the input line.
<i>commands</i>	Pointer to the <a href="#">Commands</a> struct to populate.

### 8.25.2.4 trim\_whitespace()

```
char * trim_whitespace (
    const char * line)
```

Trims leading and trailing whitespace from a given line.

#### Parameters

<i>line</i>	Pointer to the input line.
-------------	----------------------------

#### Returns

Newly allocated string with trimmed whitespace.

## 8.25.3 Variable Documentation

### 8.25.3.1 OPERATORS

```
const char* OPERATORS[] = {"&&", "||", "&", "|", ";", NULL}
```

### 8.25.3.2 REDIRECTORS

```
const char* REDIRECTORS[] = {"<<", ">>", "<", ">", NULL}
```

# Index

- add\_alias
  - aliases.h, [22](#)
- add\_to\_history
  - history.c, [46](#)
  - history.h, [30](#)
- Alias, [15](#)
  - command, [15](#)
  - name, [15](#)
  - typedef.h, [37](#)
- aliases.h
  - add\_alias, [22](#)
  - free\_aliases, [22](#)
  - get\_alias\_command, [22](#)
  - handle\_alias, [22](#)
  - handle\_unalias, [22](#)
  - is\_alias, [23](#)
  - list\_aliases, [23](#)
  - MAX\_ALIASES, [21](#)
  - remove\_alias, [23](#)
- append\_output
  - Command, [17](#)
- arg\_count
  - Command, [17](#)
- args
  - Command, [17](#)
- BackgroundProcess, [16](#)
  - count, [16](#)
  - processes, [16](#)
  - typedef.h, [37](#)
- batch.c
  - batch\_command, [40](#)
  - bg\_processes, [40](#)
  - handle\_arguments, [40](#)
  - minishell\_options, [40](#)
- batch.h
  - handle\_arguments, [25](#)
  - MAX\_ARGUMENTS, [24](#)
- batch\_command
  - batch.c, [40](#)
- bg\_processes
  - batch.c, [40](#)
  - main.c, [50](#)
- BLUE
  - typedef.h, [36](#)
- check\_bg\_processes
  - executor.c, [43](#)
  - executor.h, [27](#)
- COLOR\_RESET
  - typedef.h, [36](#)
- Command, [16](#)
  - append\_output, [17](#)
  - arg\_count, [17](#)
  - args, [17](#)
  - command, [17](#)
  - heredoc\_delimiter, [17](#)
  - input\_redirect, [17](#)
  - length, [17](#)
  - output\_redirect, [17](#)
  - typedef.h, [37](#)
- command
  - Alias, [15](#)
  - Command, [17](#)
  - Jobs, [19](#)
- command\_count
  - Commands, [18](#)
- Commands, [18](#)
  - command\_count, [18](#)
  - commands, [18](#)
  - operators, [18](#)
  - typedef.h, [37](#)
- commands
  - Commands, [18](#)
- count
  - BackgroundProcess, [16](#)
- create\_child\_process
  - executor.c, [43](#)
  - executor.h, [27](#)
- detect\_operator
  - parser.c, [51](#)
  - parser.h, [34](#)
- env.c
  - env\_vars\_head, [42](#)
  - env\_vars\_tail, [42](#)
  - get\_env\_var, [41](#)
  - print\_env\_vars, [41](#)
  - set\_env\_var, [41](#)
  - unset\_env\_var, [41](#)
- env.h
  - get\_env\_var, [26](#)
  - print\_env\_vars, [26](#)
  - set\_env\_var, [26](#)
  - unset\_env\_var, [26](#)
- env\_vars\_head
  - env.c, [42](#)
- env\_vars\_tail
  - env.c, [42](#)

- EnvVar, 18
  - name, 19
  - next, 19
  - typedef.h, 37
  - value, 19
- execute\_command
  - executor.c, 43
  - executor.h, 27
- execute\_commands
  - executor.c, 43
  - executor.h, 28
- execute\_pipes
  - executor.c, 44
  - executor.h, 28
- executor.c
  - check\_bg\_processes, 43
  - create\_child\_process, 43
  - execute\_command, 43
  - execute\_commands, 43
  - execute\_pipes, 44
  - handle\_heredoc, 44
  - internal\_cmds, 45
  - internal\_cmds\_list, 45
  - setup\_redirections, 44
- executor.h
  - check\_bg\_processes, 27
  - create\_child\_process, 27
  - execute\_command, 27
  - execute\_commands, 28
  - execute\_pipes, 28
  - handle\_heredoc, 28
  - setup\_redirections, 29
- free\_aliases
  - aliases.h, 22
- free\_commands
  - internal\_commands.c, 47
  - internal\_commands.h, 31
- free\_if\_needed
  - internal\_commands.c, 47
  - internal\_commands.h, 31
- get\_alias\_command
  - aliases.h, 22
- get\_env\_var
  - env.c, 41
  - env.h, 26
- GREEN
  - typedef.h, 36
- handle\_alias
  - aliases.h, 22
- handle\_arguments
  - batch.c, 40
  - batch.h, 25
- handle\_cd
  - internal\_commands.c, 48
  - internal\_commands.h, 32
- handle\_echo
  - internal\_commands.c, 48
  - internal\_commands.h, 32
- handle\_exit
  - internal\_commands.c, 48
  - internal\_commands.h, 32
- handle\_heredoc
  - executor.c, 44
  - executor.h, 28
- handle\_history
  - history.c, 46
  - history.h, 30
- handle\_pwd
  - internal\_commands.c, 48
  - internal\_commands.h, 32
- handle\_unalias
  - aliases.h, 22
- heredoc\_delimiter
  - Command, 17
- history.c
  - add\_to\_history, 46
  - handle\_history, 46
- history.h
  - add\_to\_history, 30
  - handle\_history, 30
- include Directory Reference, 13
- include/aliases.h, 21, 24
- include/batch.h, 24, 25
- include/env.h, 25, 26
- include/executor.h, 26, 29
- include/history.h, 30
- include/internal\_commands.h, 31, 33
- include/parser.h, 33, 35
- include/typedef.h, 36, 38
- input\_redirect
  - Command, 17
- internal\_cmds
  - executor.c, 45
- internal\_cmds\_list
  - executor.c, 45
- internal\_commands.c
  - free\_commands, 47
  - free\_if\_needed, 47
  - handle\_cd, 48
  - handle\_echo, 48
  - handle\_exit, 48
  - handle\_pwd, 48
  - work\_dir, 49
- internal\_commands.h
  - free\_commands, 31
  - free\_if\_needed, 31
  - handle\_cd, 32
  - handle\_echo, 32
  - handle\_exit, 32
  - handle\_pwd, 32
  - work\_dir, 33
- is\_alias
  - aliases.h, 23

- Jobs, [19](#)
  - command, [19](#)
  - pid, [19](#)
  - typedef.h, [37](#)
- length
  - Command, [17](#)
- list\_aliases
  - aliases.h, [23](#)
- main
  - main.c, [50](#)
- main.c
  - bg\_processes, [50](#)
  - main, [50](#)
  - parsed\_commands, [50](#)
  - write\_prompt, [50](#)
- MAX\_ALIASES
  - aliases.h, [21](#)
- MAX\_ARGUMENTS
  - batch.h, [24](#)
- MAX\_BG\_PROCESSES
  - typedef.h, [37](#)
- MAX\_COMMANDS
  - typedef.h, [37](#)
- Minishell, [1](#)
- minishell\_options
  - batch.c, [40](#)
- name
  - Alias, [15](#)
  - EnvVar, [19](#)
- next
  - EnvVar, [19](#)
- OPERATORS
  - parser.c, [52](#)
- operators
  - Commands, [18](#)
- output\_redirect
  - Command, [17](#)
- parse\_command
  - parser.c, [51](#)
  - parser.h, [34](#)
- parsed\_commands
  - main.c, [50](#)
- parser.c
  - detect\_operator, [51](#)
  - OPERATORS, [52](#)
  - parse\_command, [51](#)
  - REDIRECTORS, [52](#)
  - split\_line, [51](#)
  - trim\_whitespace, [52](#)
- parser.h
  - detect\_operator, [34](#)
  - parse\_command, [34](#)
  - split\_line, [34](#)
  - trim\_whitespace, [35](#)
- pid
  - Jobs, [19](#)
- print\_env\_vars
  - env.c, [41](#)
  - env.h, [26](#)
- processes
  - BackgroundProcess, [16](#)
- Projet - Minishell, [3](#)
- README.md, [39](#)
- REDIRECTORS
  - parser.c, [52](#)
- remove\_alias
  - aliases.h, [23](#)
- set\_env\_var
  - env.c, [41](#)
  - env.h, [26](#)
- setup\_redirections
  - executor.c, [44](#)
  - executor.h, [29](#)
- split\_line
  - parser.c, [51](#)
  - parser.h, [34](#)
- src Directory Reference, [13](#)
- src/aliases.c, [39](#)
- src/batch.c, [39](#)
- src/env.c, [40](#)
- src/executor.c, [42](#)
- src/history.c, [45](#)
- src/internal\_commands.c, [46](#)
- src/main.c, [49](#)
- src/parser.c, [50](#)
- trim\_whitespace
  - parser.c, [52](#)
  - parser.h, [35](#)
- typedef.h
  - Alias, [37](#)
  - BackgroundProcess, [37](#)
  - BLUE, [36](#)
  - COLOR\_RESET, [36](#)
  - Command, [37](#)
  - Commands, [37](#)
  - EnvVar, [37](#)
  - GREEN, [36](#)
  - Jobs, [37](#)
  - MAX\_BG\_PROCESSES, [37](#)
  - MAX\_COMMANDS, [37](#)
- unset\_env\_var
  - env.c, [41](#)
  - env.h, [26](#)
- value
  - EnvVar, [19](#)
- work\_dir
  - internal\_commands.c, [49](#)
  - internal\_commands.h, [33](#)

write\_prompt  
main.c, [50](#)