Minishell by Gunjan Dhanuka

1.0.0

Generated by Doxygen 1.8.18

Chapter 1

File Index

1.1 File List

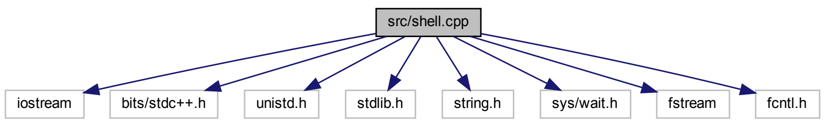
Here is a list of all files with brief descriptions:

Chapter 2

File Documentation

2.1 src/shell.cpp File Reference

```
#include <iostream>
#include <bits/stdc++.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>
#include <fstream>
#include <fcntl.h>
```
Include dependency graph for shell.cpp:



Macros

- #define TOK_BUFFER_SIZE 64
- #define READ_LINE_BUFFER_SIZE 1024
- #define TOK_DELIM " =\t\r\n\a"
- #define ZERO 0
- #define ONE 1
- #define NEWLINE '\n'
- #define MAXCOMS 100
- #define err(x) cerr << red << x << reset << endl;

Functions

- const std::string red ("\033[0;31m")
- const std::string green ("\033[0;32m")
- const std::string blue ("\033[0;34m")
- const std::string yellow ("\033[0;33m")
- const std::string cyan ("\033[0;36m")
- const std::string reset ("\033[0m")
- int number_of_builtins ()
- int my_cd (char **args)
- int my_help (char **args)
- int my_exit (char **args)
- int my_printenv (char **args)
- int my_setenv (char **args)
- int my_unsetenv (char **args)
- int my_history (char **args)
- int my_echo (char **args)
- void printCurrentDir ()
- int read_input (char *s)
- char ** split_input (char *s)
- void parser (char *s, char **args)
- int launcher (char **args)
- void runsource (int pfd[ ], char **args)
- void rundest (int pfd[ ], char **args)
- int pipeLauncher (char **parsed, char **piped)
- int execute (char **args)
- int inbuiltHandler (char **args)
- int parserPipe (char *str, char **pipedarr)
- int redirection (char **args)
- int isRedirection (char **args)
- int processInput (char *str, char **parsed, char **piped)
- void main_process ()
- int main ()

Variables

- const char * builtin_str [ ]

2.1.1 Detailed Description

CS224 Assignment 2: Implementing a mini-shell using C/C++

Author

Gunjan Dhanuka

Version

1.0 09/2021

2.1.2 Macro Definition Documentation

2.1.2.1 err

```
#define err(
              x ) cerr << red << x << reset << endl;
```

Definition at line 25 of file shell.cpp.

2.1.2.2 MAXCOMS

```
#define MAXCOMS 100
```

Definition at line 24 of file shell.cpp.

2.1.2.3 NEWLINE

```
#define NEWLINE '\n'
```

Definition at line 23 of file shell.cpp.

2.1.2.4 ONE

```
#define ONE 1
```

Definition at line 22 of file shell.cpp.

2.1.2.5 READ_LINE_BUFFER_SIZE

```
#define READ_LINE_BUFFER_SIZE 1024
```

Definition at line 19 of file shell.cpp.

2.1.2.6 TOK_BUFFER_SIZE

```
#define TOK_BUFFER_SIZE 64
```

Definition at line 18 of file shell.cpp.

2.1.2.7 TOK_DELIM

```
#define TOK_DELIM " =\t\r\n\a"
```

Definition at line 20 of file shell.cpp.

2.1.2.8 ZERO

```
#define ZERO 0
```

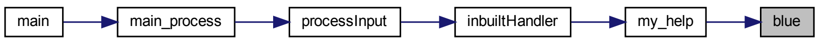
Definition at line 21 of file shell.cpp.

2.1.3 Function Documentation

2.1.3.1 blue()

```
const std::string blue (
             "\033[0;34m"  )
```

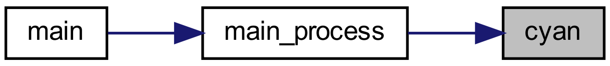
Here is the caller graph for this function:



2.1.3.2 cyan()

```
const std::string cyan (
            "\033[0;36m" )
```

Here is the caller graph for this function:



2.1.3.3 execute()

```
int execute (
            char ** args )
```

Executes a shell-command which is not an inbuilt/piped command.

Parameters

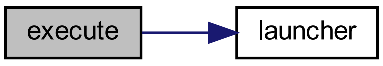
args	the array containing the command and its args

Returns

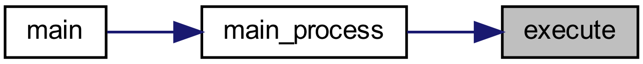
one if the command is empty; else the output of launcher function

Definition at line 498 of file shell.cpp.

Here is the call graph for this function:



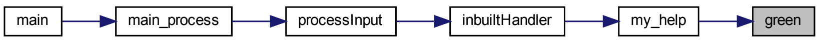
Here is the caller graph for this function:



2.1.3.4 green()

```
const std::string green (
             "\033[0;32m"  )
```

Here is the caller graph for this function:



2.1.3.5 inbuiltHandler()

```
int inbuiltHandler (
             char ** args )
```

Handles the inbuilt commands of the minishell. If the command is present, it is executed.

Parameters

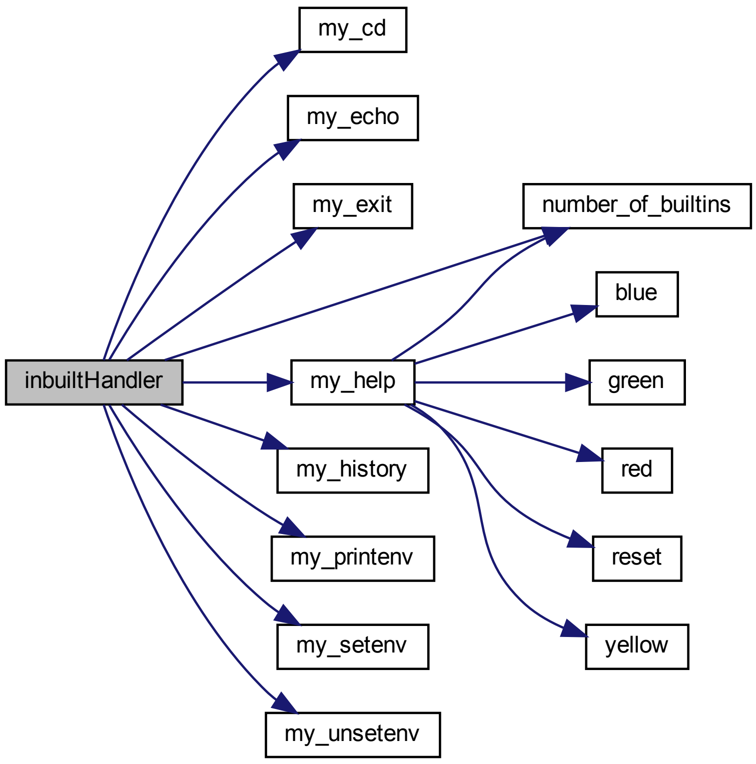
args	the array containing the command and its args

Returns

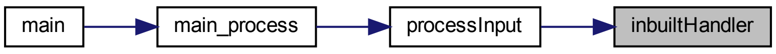
one if the command is present and successfully executed, zero if command not present

Definition at line 515 of file shell.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



2.1.3.6 isRedirection()

```
int isRedirection (
            char ** args )
```

Checks if the user input args contain a redirection operator.

Parameters

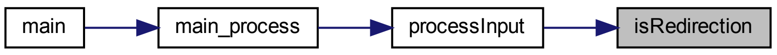
args	the array containing the command and its args

Returns

one if any of the redirection operators are present, zero otherwise.

Definition at line 689 of file shell.cpp.

Here is the caller graph for this function:



2.1.3.7 launcher()

```
int launcher (
            char ** args )
```

Launches the process using the args supplied. Executes the parent and then waits for the child to finish.

Parameters

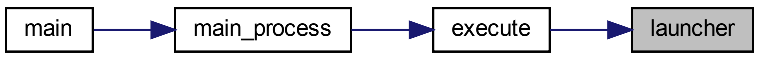
args	the string array containing the parsed tokens

Returns

one if the process completed successfully

Definition at line 381 of file shell.cpp.

Here is the caller graph for this function:



2.1.3.8 main()

```
int main ( )
```

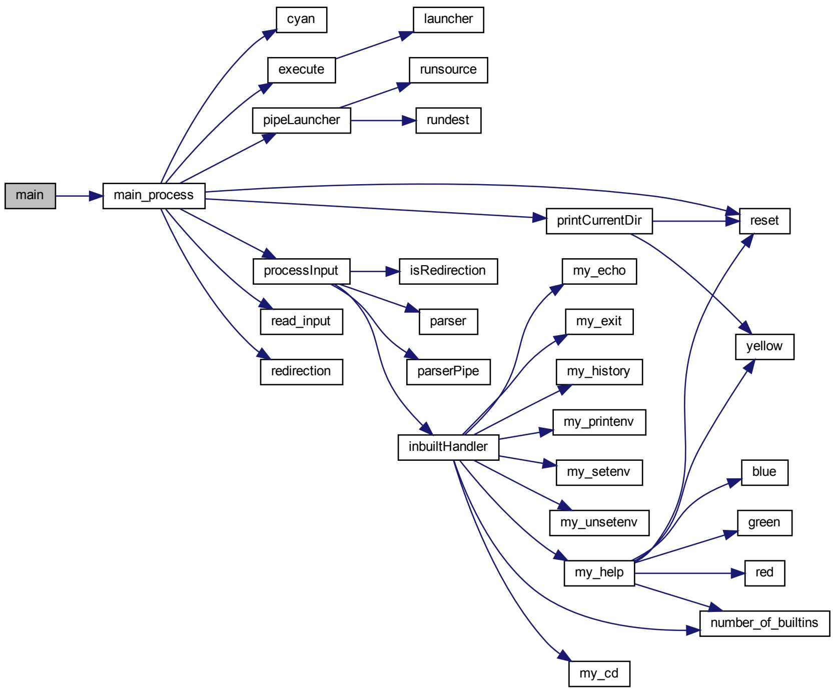
The main function of a C++ program.

Returns

zero if exits normally

Definition at line 809 of file shell.cpp.

Here is the call graph for this function:



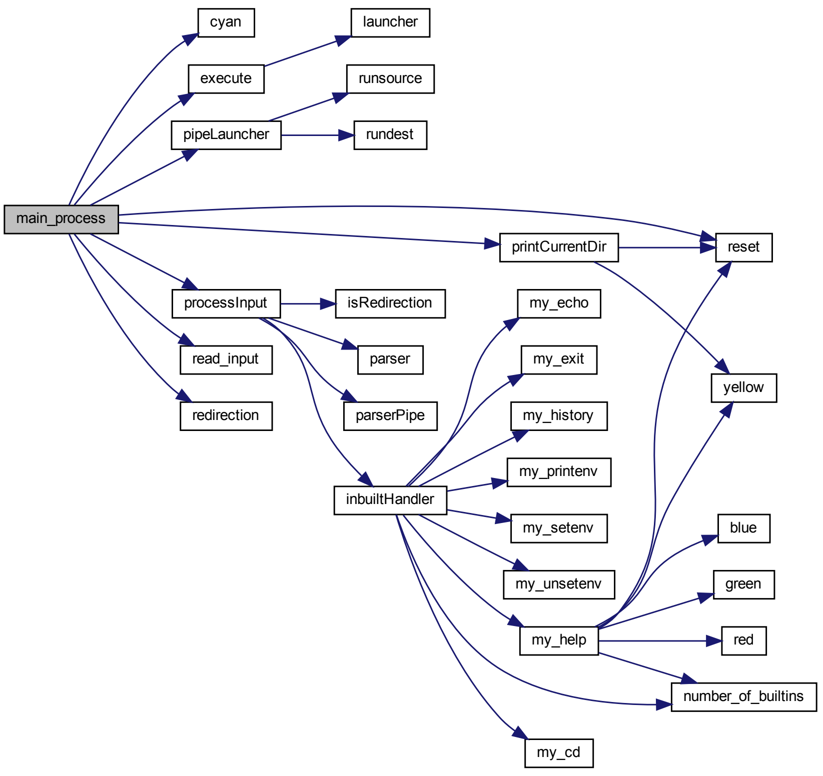
2.1.3.9 main_process()

```
void main_process ( )
```

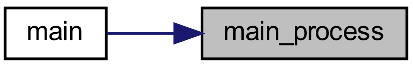
Driver function, contains the main infinite loop of the terminal and handles the execution of different functions from reading input to running commands.

Definition at line 762 of file shell.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



2.1.3.10 my_cd()

```
int my_cd (
              char ** args )
```

Changes the directory using chdir

Parameters

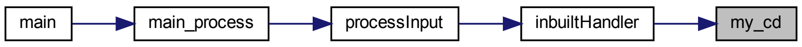
args	

Returns

one if executed correctly, zero if some error encountered!

Definition at line 62 of file shell.cpp.

Here is the caller graph for this function:



2.1.3.11 my_echo()

```
int my_echo (
              char ** args )
```

A re-implementation of the 'echo' command of UNIX terminal.

Parameters

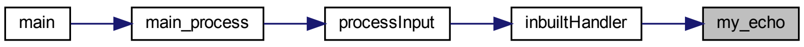
args	

Returns

one if executed correctly, zero if some error encountered!

Definition at line 199 of file shell.cpp.

Here is the caller graph for this function:



2.1.3.12 my_exit()

```
int my_exit (
             char ** args )
```

Exits from the shell

Parameters

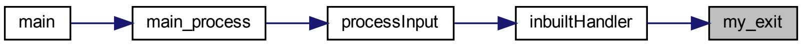
args	

Returns

zero for exit

Definition at line 108 of file shell.cpp.

Here is the caller graph for this function:



2.1.3.13 my_help()

```
int my_help (
             char ** args )
```

Prints info about the shell and its functionalities

Parameters

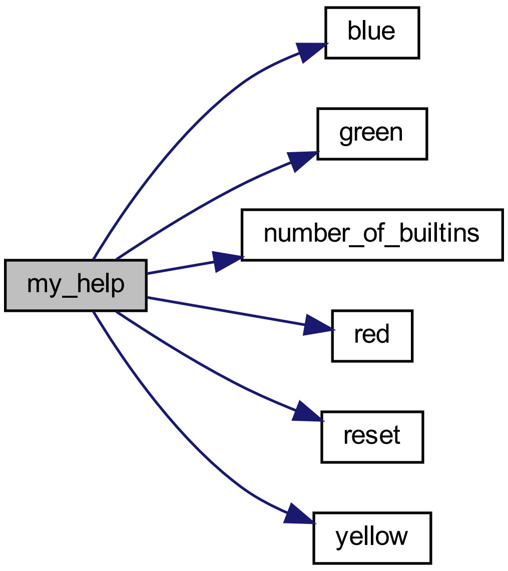
args	

Returns

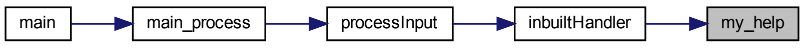
one if executed correctly, zero if some error encountered!

Definition at line 86 of file shell.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



2.1.3.14 my_history()

```
int my_history (
            char ** args )
```

Prints the history of the shell so far since the first execution.

Parameters

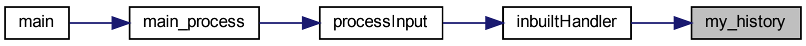
args	

Returns

one if executed correctly, zero if some error encountered!

Definition at line 183 of file shell.cpp.

Here is the caller graph for this function:



2.1.3.15 my_printenv()

```
int my_printenv (
             char ** args )
```

Prints the environment variable asked for by the `args`; if no argument is passed, then it prints the five common env varibles!

Parameters

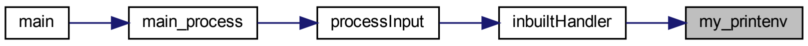
args	

Returns

one if executed correctly, zero if some error encountered!

Definition at line 119 of file shell.cpp.

Here is the caller graph for this function:



2.1.3.16 my_setenv()

```
int my_setenv (
             char ** args )
```

Changes the value of preexisting env variable or creates a new one.

Parameters

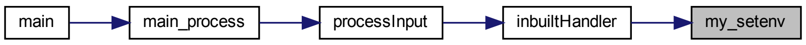
args	

Returns

one if executed correctly, zero if some error encountered!

Definition at line 148 of file shell.cpp.

Here is the caller graph for this function:



2.1.3.17 my_unsetenv()

```
int my_unsetenv (
            char ** args )
```

Removes the value of the environment varible

Parameters

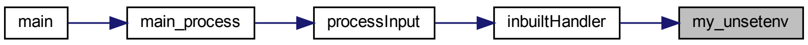
args	

Returns

one if executed correctly, zero if some error encountered!

Definition at line 165 of file shell.cpp.

Here is the caller graph for this function:



2.1.3.18 number_of_builtins()

```
int number_of_builtins ( )
```

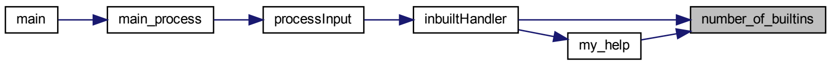
Tells the number of built-in functions of the shell

Returns

number of built-in functions

Definition at line 50 of file shell.cpp.

Here is the caller graph for this function:



2.1.3.19 parser()

```
void parser (
             char * s,
             char ** args )
```

Changes the directory using chdir

Parameters

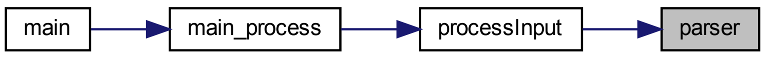
s	the string to be parsed
args	the string array to store the parsed tokens

Returns

none

Definition at line 356 of file shell.cpp.

Here is the caller graph for this function:



2.1.3.20 parserPipe()

```
int parserPipe (
             char * str,
             char ** pipedarr )
```

Parses the user input that contains a pipe symbol. Separates into pre-pipe and post-pipe args.

Parameters

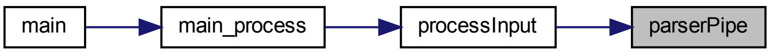
str	the string input by the user
pipedarr	the array which will store the args before and after pipe separately.

Returns

one if pipe is present and zero if no piping present.

Definition at line 582 of file shell.cpp.

Here is the caller graph for this function:



2.1.3.21 pipeLauncher()

```
int pipeLauncher (
             char ** parsed,
             char ** piped )
```

Starts the pipe and then calls the runsource and rundest in order to complete the piping process.

Parameters

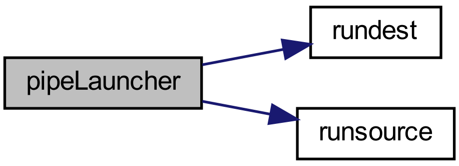
parsed	the arguments before the pipe(|) symbol
piped	the arguments after the pipe(|) symbol

Returns

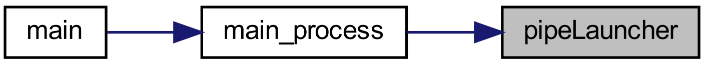
one if the process completed successfully

Definition at line 473 of file shell.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



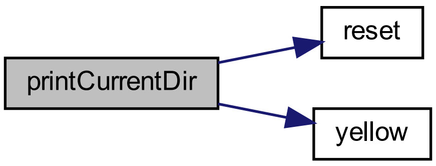
2.1.3.22 printCurrentDir()

```
void printCurrentDir ( )
```

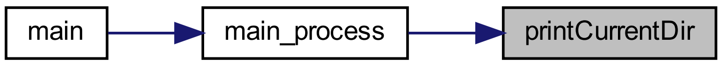
Prints the current working directory ahead of the shell-prompt.

Definition at line 233 of file shell.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



2.1.3.23 processInput()

```
int processInput (
             char * str,
             char ** parsed,
             char ** piped )
```

Handles the inbuilt commands of the minishell. If the command is present, it is executed.

Parameters

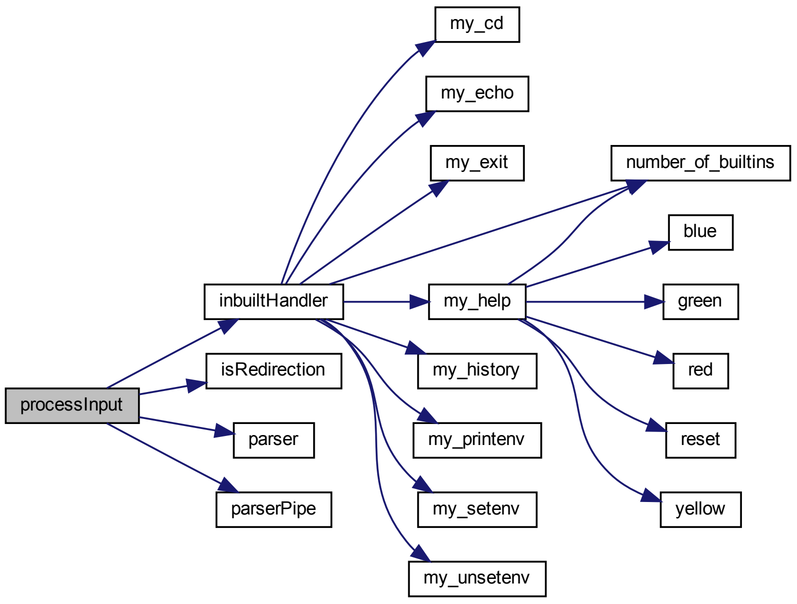
str	containing the input given by the user
parsed	string array to store the commands and the args before pipe, if any.
piped	string array to store the commands and its args after the pipe, if any.

Returns

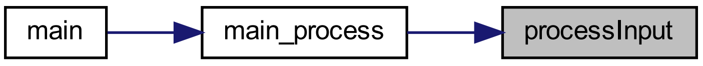
zero if the command is inbuilt, one if the command is usual shell command, two if the user seeks piping, and three if the user seeks redirection.

Definition at line 722 of file shell.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



2.1.3.24 read_input()

```
int read_input (
                char * s )
```

Reads the input from the shell given by the user using the old-school buffer mothod just for working closer to the terminal.

Parameters

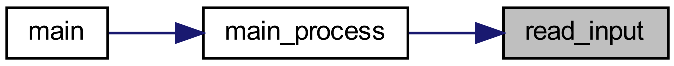
s	the string containing user input

Returns

one if the input string was empty

Definition at line 246 of file shell.cpp.

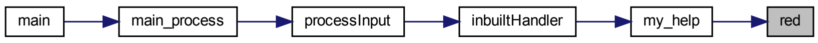
Here is the caller graph for this function:



2.1.3.25 red()

```
const std::string red (
             "\033[0;31m"  )
```

Here is the caller graph for this function:



2.1.3.26 redirection()

```
int redirection (
             char ** args )
```

Performs redirection of one or multiple of three kinds (<, >, >>) using file descriptors that duplicate the input/output file to stdin/stdout.

Parameters

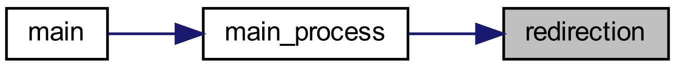
args	the array containing the command and its args

Returns

zero if executed successfully.

Definition at line 606 of file shell.cpp.

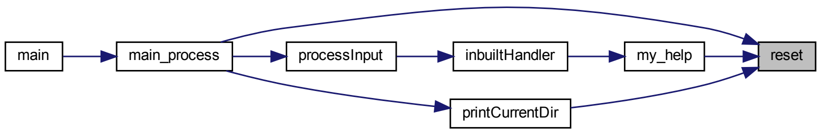
Here is the caller graph for this function:



2.1.3.27 reset()

```
const std::string reset (
             "\033[0m"  )
```

Here is the caller graph for this function:



2.1.3.28 rundest()

```
void rundest (
             int pfd[],
             char ** args )
```

Runs the second command in a piped statement.

Parameters

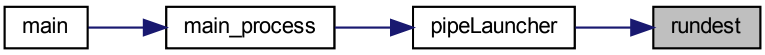
pfd	the array containing the file descriptors for read and write of the pipe
args	the string array containing the parsed tokens

Returns

none

Definition at line 448 of file shell.cpp.

Here is the caller graph for this function:



2.1.3.29 runsource()

```
void runsource (
            int pfd[],
            char ** args )
```

Runs the first command in a piped statement.

Parameters

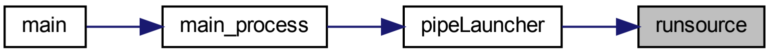
pfd	the array containing the file descriptors for read and write of the pipe
args	the string array containing the parsed tokens

Returns

none

Definition at line 423 of file shell.cpp.

Here is the caller graph for this function:



2.1.3.30 split_input()

```
char** split_input (
             char * s )
```

Splits the input into different tokens separated by pre-defined delimiters.

Parameters

s	the string containing user input

Returns

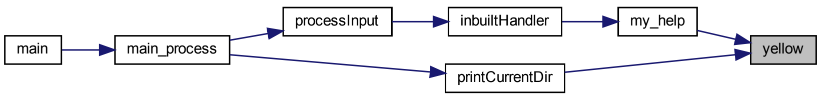
array of strings containing the tokens

Definition at line 312 of file shell.cpp.

2.1.3.31 yellow()

```
const std::string yellow (
             "\033[0;33m"  )
```

Here is the caller graph for this function:



2.1.4 Variable Documentation

2.1.4.1 builtin_str

```
const char* builtin_str[]
```

Initial value:

```
= {
    "cd",
    "help",
    "exit",
    "printenv",
    "setenv",
    "unsetenv",
    "history",
    "echo"}
```

Definition at line 36 of file shell.cpp.

Index