

utils.c


strjoin_with_free

```
if (!*s1)
    malloc(strlen(s2) + 1)
else
    malloc(strlen(s1)+strlen(s2)+1)
s1 を copy
s2 を copy
最後に '\0'
if (select = FREE_S1)
    free(s1)
if (select = FREE_S2)
    free(s)
return (result)
```

free.c

free_env

```
while(env)
    if(env->key)
        free(env->key)
    if(env->key)
        free(env->key)
    free(env)
    if (env->next)
        env = env->next
```



free_split

```
while(split[i])
    free(split[i++])
free(split)
```

free_cmd

```
if (cmd->pathname)
    free(cmd->pathname)
if (cmd->cmd)
    free_split(cmd->cmd)
if (cmd->err_msg)
    free(cmd->err_msg)
free(cmd)
```


process.c

t_cmd

```
int readfd
int writefd
int pp[2]
char *pathname
char **cmd
char *err_msg
```

run_process

```
make_token_lst(line)
while(++i < cmd_count)
    make_cmd
    (l か END) まで token++
    make_fork
    child_prcess
    parent_process
    token_lstclear
    return (wait_process)
```

wait_process

```
while (++i < cmd_count)
    waitpid
    WEXITED
    return (exit_status)
```

make_token_lst

cmd_count

make_cmd // コマンドを作る

make_fork

child_process

```
if (err_msg)
    exit_child_process
if (readfd)
    dup2(readfd, 0)
if (writefd)
    dup2(writefd, 1)
else if (i != cmd_count)
    dup2(pp[1], 1)
close_fds
execve // コマンド実行
exit
```

parent_process

```
if (i != cmd_count - 1)
    dup2(pp[0], 0)
close_fds
ft_free_cmd(cmd)
```

token_lstclear

make_token_lst.c

command.c

process_utils.c

process_utils.c

process_utils.c

process_utils.c

free.c

process_utils.c

process_utils.c

make_fork

```
fork
```

cmd_count

pipe(l) の数 +1 を返す

close_fds

```
while (token)
    (unlink(FILE_NAME))
    free(token->word)
    free(token)
```

exit_child_process

```
if (err_msg)
    ft_print(err_msg)
close_fds
exit
```

close_fds

```
close(readfd)
close(writefd)
close(pp[0])
close(pp[1])
```


make_token_lst.c

t_token

```
char *word
t_kind kind
s_token *next
s_token *pre
```

make_token_lst

```
ft_split(line, ' ')
while(split[++i])
    lstnew(split[i])
add_token_kind
ft_free_split
free(line)
return (token)
```

ft_split

libft

static lstnew

```
malloc(*t_token)
word = ft_strdup(split[i])
kind = -1
next = NULL
lst_add_back
```

static lst_add_back

static lst_last

add_token_kind

add_token_kind.c

ft_free_split

utils.c

add_token_kind.c

e_kind

```
PIPE
COMMAND
ARGUMENT
SKIP
RDFILE
WRFILE
WRFILE_APP
LIMITTER
END
```

add_token_kind

while (token)

if(!ft_memcmp(token, " |", 2))

kind = PIPE token++

else if(!ft_memcmp(token, " <", 2))

kind = SKIP

next->kind = RDFILE token++

else if(!ft_memcmp(token, " <<", 3))

kind = SKIP

next->kind = LIMITTER token++

else if(!ft_memcmp(token, " >", 2))

kind = SKIP

next->kind = WRFILE token++

else if(!ft_memcmp(token, " >>", 3))

kind = SKIP

next->kind = WRFILE_APP token++

else

if(token->pre != COMMAND)

kind = COMMAND

if (token->pre == COMMAND)

kind = ARGUMENT

token++ or break

command.c ①

make_cmd

```
malloc(*t_cmd)
init_cmd
getenv_str(env, "PATH" )
ft_split(↑, ':' )
while(token)
    if (kind = PIPE)
        safe_pipe
        token++
        break
    else if (kind = RDFILE || LIMITTER)
        open_read_file
    else if (kind = WRFILE || WRFILE_APP)
        open_write_file
    else if (kind = COMMAND)
        make_path_and_cmd
        while(token->next)
            if (next->kind = OPTION)
                token++
            else
                break
ft_free_split(path)
if (access(pathname))
    set_err_message
return(cmd)
```

t_cmd

```
int    readfd
int    writefd
int    pp[2]
char   *pathname
char   **cmd
char   *err_msg
```

init_cmd

getenv_str

ft_split // PATH の split

safe_pipe

open_read_file

open_write_file

make_path_and_cmd

ft_free_split

command_utils.c

libft

command_utils.c

open_file.c

command.c ②

free.c

set_err_message

```
if (err_msg)
    return
if (!str || !*str)
    strjoin_with_free(" Commnad ' ' not found" )
else
    strjoin_with_free(" Bash: ", str)
    strjoin_with_free(err_msg, "command not found" )
```


command.c ②

make_path_and_cmd

```
getenv_str(env, "PWD" )
make_cmd_array
if (cmd[0])
    if (cmd[0] == '/' )
        strjoin_with_free("", cmd[0])
    if (cmd[0] == '.' )
        make_pwd_path
    else
        make_cmd_and_check_access
if (!pathname)
    strjoin_with_free("", cmd[0])
```

getenv_str

command_utils.c

str_join_with_free

utils.c

make_pwd_path

command_utils.c

make_cmd_and_check_access

```
if (!path) // unset PATH の時
    return (make_pwd_path)
while (path[++i])
    strjoin_with_free(path[i], "/" )
    strjoin_with_free(str, command)
    if (!access) // 実行可能な時
        return (str)
    free(str)
return (strjoin_with_free(" x" , command))
```

str_join_with_free

utils.c

make_cmd_array

```
while (kind == COMMAND || OPTION)
    count++ // コマンドとオプションの数
    if (ptr->next)
        ptr++
    else
        break
malloc((char *) * (count + 1))
while (++i < count)
    cmd[i] = ft_strdup(word)
    token++
cmd[i] = NULL
return (cmd)
```

ft_strdup

libft

command_utils.c

init_cmd

safe_pipe

make_pwd_path

strjoin_with_free // pwd + / + cmd[0]

str_join_with_free

utils.c

getenv_str

if (!env)

return (NULL)

while (env->key)

if (!ft_memcmp(str, env->key))

return (key->value)

env++

return (NULL)

ft_memcmp

libft

open_file.c

heredoc_process

```
open(FILENAME)
while (1)
    ft_printf(1, " h> ")
    get_next_line(0)
    print_limiter_warning(eof)
    break
    if (ft_memcmp(str, eof) = 10)
        free(str)
        break
    ft_printf(fd, "%s", str)
    free(str)
    str = NULL
return (close(fd), TRUE)
```

ft_printf

libft

get_next_line

libft

print_limiter_warning

ft_memcmp

libft

open_read_file

```
if (kind = LIMITTER)
    if (heredoc_process = FALSE)
        return
    open(FILE_NAME, O_RDONLY)
    if (readfd < 0)
        set_file_err
    if (kind = RDFILE)
        open(word, O_RDONLY)
        if (readfd < 0)
            set_file_err
```

set_file_err

set_file_err

open_write_file

```
if (kind = WRFILE_APP)
    open(word, CREAT/RDWR/APP)
    if (writefd < 0 && !err_msg)
        set_file_err
    if (kind = WRFILE)
        open(word, CREAT/RDWR/TRUNC)
        if (writefd < 0 && !err_msg)
            set_file_err
```

set_file_err

set_file_err

set_file_err

```
str_join_with_free
    "bash: " + filename
    ↑ + ": "
    ↑ + err_msg
    ↑ + "\n"
return (message)
```

utils.c