🚀 Minishell - Kapsamlı Dokümantasyon

📋 İçindekiler

🎯 Proje Genel Bakış

Minishell, bash shell'inin basit bir implementasyonudur. Bu proje, shell'in temel işlevlerini yerine getirerek kullanıcıların komut satırı deneyimi yaşamalarını sağlar.

🔧 Temel Özellikler

- **Komut Çalıştırma**: Dış komutlar ve builtin komutlar
- **Pipe İşlemleri**: Birden fazla komutun bağlanması (`|`)
- **Yönlendirme**: Input/Output yönlendirmeleri (`<`, `>`, `>>`)
- **Heredoc**: Çoklu satır giriş (`<<`)
- **Değişken Genişletme**: Environment variable expansion (`$VAR`)
- **Quote İşleme**: Tek ve çift tırnak desteği
- **Signal Handling**: Ctrl+C, Ctrl+Z gibi sinyaller
- **Builtin Komutlar**: echo, cd, pwd, export, unset, env, exit

---

🔄 Algoritma ve İşleyiş Yapısı

Minishell'in çalışma akışı şu aşamalardan oluşur:

1. 📥 INPUT ALMA

```
📋 Readline ile kullanıcıdan komut alınır
📋 History'ye eklenir
📋 Boş input kontrolü yapılır
```

2. 🔤 TOKENIZATION (Lexical Analysis)

```
📋 Input string parçalara ayrılır
📋 Operatörler, kelimeler ve quote'lar tanımlanır
📋 Token array'i oluşturulur
```

3. 🔍 EXPANSION (Variable Expansion)

```
📋 $VAR formatındaki değişkenler genişletilir
📋 Quote context'ine göre işlem yapılır
📋 Exit status ($?) değişkeni işlenir
```

4. 📝 PARSING (Syntax Analysis)

```
📋 Token'lar komut yapılarına dönüştürülür
📋 Pipe'lar ve yönlendirmeler işlenir
📋 Command list oluşturulur
```

5. ⚡ EXECUTION (Command Execution)

```
📋 Builtin vs external komut kontrolü
📋 Process management (fork/exec)
📋 Pipe setup ve data transfer
📋 Redirection handling
```

6. 🔚 CLEANUP & EXIT

```
📋 Memory cleanup
📋 Process waiting
📋 Exit status setting
```

🏗️ Modül Yapıları

📦 MAIN MODUL (main/)

Sorumluluğu: Ana program döngüsü ve setup işlemleri

```c
// minishell.c - Ana program döngüsü
int main(int ac, char **av, char **env)
{
    t_req res = setup(av, env);
    while (process_main_loop(&res))
        ;
    return (res.exit_stat);
}
```

🔤 TOKENIZER MODÜLÜ (src/tokenizer/)

Sorumluluğu: Input string'i token'lara ayırma

```
// Tokenization süreci:
Input: "echo 'hello world' | grep hello"
↓
Tokens: ["echo", "'hello world'", "|", "grep", "hello"]
```

Anahtar Fonksiyonlar:

- `tokenize_input()`: Ana tokenization fonksiyonu
- `get_word_token()`: Kelime token'ları oluşturur
- `get_operator_token()`: Operatör token'ları oluşturur

🔍 EXPANDER MODÜLÜ (src/expander/)

Sorumluluğu: Variable expansion işlemleri

```
// Expansion örneği:
Input: "echo $HOME/file"
↓
Expanded: "echo /Users/username/file"
```

📝 PARSER MODÜLÜ (src/parser/)

Sorumluluğu: Token'ları komut yapılarına dönüştürme

```
// Parser yapısı:
Token Array → Command List (t_list of t_shell)
```

Anahtar Yapılar:

```
typedef struct s_shell {
    char    **full_cmd;     // Komut ve argümanları
    char    *full_path;     // Komutun tam yolu
    int     infile;         // Input file descriptor
    int     outfile;        // Output file descriptor
    t_redirect *redirects;  // Yönlendirme listesi
} t_shell;
```

⚡ EXECUTOR MODÜLÜ (src/executor/)

Sorumluluğu: Komutları çalıştırma ve process yönetimi

Pipeline İşleyişi:

```
cmd1 | cmd2 | cmd3
  ↓     ↓     ↓
[fd0] [fd1] [fd2]  ← Input FDs
  ↓     ↓     ↓
[pid1][pid2][pid3] ← Child processes
  ↓     ↓     ↓
[pipe][pipe][out]  ← Output destinations
```

🔧 BUILTIN MODÜLÜ (src/builtin/)

Sorumluluğu: Yerleşik komutların implementasyonu

Desteklenen Builtin'ler:

- `echo`: Metin yazdırma (flags: -n)
- `cd`: Dizin değiştirme
- `pwd`: Mevcut dizin gösterme
- `export`: Environment variable tanımlama
- `unset`: Environment variable silme
- `env`: Environment variable'ları listeleme
- `exit`: Shell'den çıkış

☑ Örnekler - Doğru Çalışma

◇ ÖRNEK 1: PIPE YoĞUN KULLANIM

```
# Komut
$ env | sort | grep -v SHLVL | head -5

# İşleyiş
┌─────────────────────────────────────────────────────┐
│ 1. TOKENIZATION                                     │
│    ["env", "|", "sort", "|", "grep", "-v", "SHLVL",    │
│     "|", "head", "-5"]                              │
└─────────────────────────────────────────────────────┘

┌─────────────────────────────────────────────────────┐
│ 2. PARSING                                          │
│    cmd1: env                                        │
│    cmd2: sort                                       │
│    cmd3: grep -v SHLVL                              │
│    cmd4: head -5                                    │
└─────────────────────────────────────────────────────┘

┌─────────────────────────────────────────────────────┐
│ 3. EXECUTION (Pipeline)                             │
│    pipe1: env → sort                                │
│    pipe2: sort → grep                               │
```

```
│    pipe3: grep → head                                          │
│    Final: head → stdout                                        │
└────────────────────────────────────────────────────────────────┘

# Sonuç
HOME=/Users/username
LANG=en_US.UTF-8
PATH=/usr/local/bin:/usr/bin:/bin
PWD=/current/directory
USER=username
```

◇ ÖRNEK 2: REDIRECTION YoĞUN KULLANIM

```
# Komut
$ cat < input.txt | grep "pattern" > output.txt 2>> error.log

# İşleyiş
┌────────────────────────────────────────────────────────────────┐
│ 1. REDIRECTION PARSING                                         │
│    cmd1: cat                                                   │
│      - Input: < input.txt                                      │
│    cmd2: grep "pattern"                                        │
│      - Output: > output.txt                                    │
│      - Error: 2>> error.log                                    │
└────────────────────────────────────────────────────────────────┘

┌────────────────────────────────────────────────────────────────┐
│ 2. FILE DESCRIPTOR SETUP                                       │
│    cat:                                                        │
│      stdin = open("input.txt", O_RDONLY)                       │
│      stdout = pipe_fd[1]                                       │
│    grep:                                                       │
│      stdin = pipe_fd[0]                                        │
│      stdout = open("output.txt", O_WRONLY|O_CREAT)             │
│      stderr = open("error.log", O_WRONLY|O_APPEND)             │
└────────────────────────────────────────────────────────────────┘

┌────────────────────────────────────────────────────────────────┐
│ 3. PROCESS EXECUTION                                           │
│    fork() → cat process                                        │
│    fork() → grep process                                       │
│    waitpid() → her iki process için                            │
└────────────────────────────────────────────────────────────────┘
```

◇ ÖRNEK 3: DEFAULT (Genel Kullanım)

```
# Komut
$ export NAME="John Doe" && echo "Hello $NAME" && pwd

# İşleyiş
┌────────────────────────────────────────────────────────────────┐
```

```
| 1. COMMAND SEPARATION (&&)                                  |
|    cmd1: export NAME="John Doe"                             |
|    cmd2: echo "Hello $NAME"                                 |
|    cmd3: pwd                                                |
└─────────────────────────────────────────────────────────────┘

┌─────────────────────────────────────────────────────────────┐
| 2. EXECUTION SEQUENCE                                       |
|    Step 1: export NAME="John Doe"                           |
|            → builtin, envp güncellenir                      |
|            → exit_status = 0                                |
|    Step 2: echo "Hello $NAME"                               |
|            → $NAME expansion → "Hello John Doe"             |
|            → builtin echo çalıştırılır                      |
|    Step 3: pwd                                              |
|            → builtin pwd çalıştırılır                       |
└─────────────────────────────────────────────────────────────┘


# Sonuç
Hello John Doe
/current/working/directory
```

❌ Hatalı Örnekler

🟤 ÖRNEK 1: Syntax Hatası - Pipe

```
# Hatalı Komut
$ echo "hello" | | grep hello
           ↑
      Syntax error: unexpected token '|'

# İşleyiş
┌─────────────────────────────────────────────────────────────┐
| TOKENIZATION: ["echo", "hello", "|", "|", "grep", "hello"] |
| PARSING: pipe after pipe detected                           |
| ERROR: ms_error(ERR_PIPE_SYNTAX, "|", 2, req)               |
| EXIT_STATUS: 2                                              |
└─────────────────────────────────────────────────────────────┘
```

🟤 ÖRNEK 2: Redirection Hatası

```
# Hatalı Komut
$ cat < nonexistent.txt
         ↑
    No such file or directory

# İşleyiş
┌─────────────────────────────────────────────────────────────┐
| PARSING: input redirection to "nonexistent.txt"             |
```

```
| EXECUTION: open("nonexistent.txt", O_RDONLY) fails      |
| ERROR: perror("minishell: nonexistent.txt")             |
| EXIT_STATUS: 1                                          |
└─────────────────────────────────────────────────────────┘
```

Önemli Kod Notları

MEMORY MANAGEMENT

```c
// Kritik memory cleanup noktaları:
void free_tokens(t_token **tokens) {
    // Token array'lerinin temizlenmesi
}

void free_cmds(t_list *cmds) {
    // Command list'inin temizlenmesi
}

// should_exit flag kullanımı:
if (req->should_exit) {
    return (0);  // exit() yerine return kullan
}
```

SIGNAL HANDLING

```c
// Heredoc için özel signal handler:
static void heredoc_sigint_handler(int sig) {
    (void)sig;
    write(1, "\n", 1);
    _exit(130);  // Bash-compatible exit code
}

// Ana program signal setup:
void setup_signals(void) {
    signal(SIGINT, sigint_handler);
    signal(SIGQUIT, SIG_IGN);
}
```

PROCESS MANAGEMENT

```c
// Pipeline execution pattern:
pid_t *pids = malloc(sizeof(pid_t) * cmd_count);
for each command:
    pids[i] = fork();
    if (pids[i] == 0) {
```

```
            // Child process - command execution
            execve(cmd->full_path, cmd->full_cmd, envp);
        }

    // Wait for all processes:
    for (i = 0; i < cmd_count; i++) {
        waitpid(pids[i], &status, 0);
        if (WIFEXITED(status))
            req->exit_stat = WEXITSTATUS(status);
    }
```

◇ PIPE IMPLEMENTATION

```
// Pipe kurulumu:
int pipe_fd[2];
pipe(pipe_fd);

// First command output → pipe input
dup2(pipe_fd[1], STDOUT_FILENO);
close(pipe_fd[1]);

// Second command input ← pipe output
dup2(pipe_fd[0], STDIN_FILENO);
close(pipe_fd[0]);
```

◇ TOKENIZER LOGIC

```
// Quote handling kritik mantık:
if (input[i] == '\'' || input[i] == '"') {
    quote_char = input[i];
    i++; // Skip opening quote
    while (input[i] && input[i] != quote_char) {
        // Quote içindeki karakterleri al
        append_char(result, input[i++]);
    }
    if (input[i] == quote_char) i++; // Skip closing quote
}
```

◇ EXPANSION ALGORITHM

```
// Variable expansion algoritması:
if (input[i] == '$') {
    i++; // Skip $
    if (input[i] == '?') {
        // Exit status expansion
        expanded = ft_itoa(req->exit_stat);
```

```
        } else {
            // Normal variable expansion
            var_name = extract_var_name(input, &i);
            expanded = mini_getenv(var_name, envp);
        }
        result = ft_strjoin(result, expanded);
    }
```

◇ 42 NORM COMPLIANCE

```
// Fonksiyon uzunluğu limiti (25 satır):
static int handle_complex_logic(args) {
    // Karmaşık mantık küçük fonksiyonlara bölünür
}

// Global variable yasağı:
// Signal handling için struct geçişi:
typedef struct s_request {
    // Global yerine struct member kullanımı
    int should_exit;
    int exit_stat;
} t_req;
```

SONUÇ

Bu minishell implementasyonu, bash'in temel özelliklerini destekleyerek:

- ☑ Memory leak-free çalışma
- ☑ Bash-compatible exit codes
- ☑ 42 norm compliance
- ☑ Signal handling (Ctrl+C support)
- ☑ Pipe ve redirection desteği
- ☑ Builtin command implementasyonu

sağlayarak tam fonksiyonel bir shell deneyimi sunmaktadır.

Dokümantasyon Tarihi: 27 Temmuz 2025 Versiyon: 1.0 Yazarlar: haloztur, musoysal