# 第三次上机作业

## 软件2306班 曹佳宁 20232241334

### 什么是命令行解释器

- 也称为命令行界面（CLI）或终端模拟器，是一种允许用户通过文本输入指令与计算机操作系统交互的软件。它接收用户输入的命令，然后解析并执行这些命令，通常用于执行系统级任务、管理文件、运行程序等。在命令行解释器中，用户不需要图形用户界面（GUI），而是直接键入特定的命令和参数来完成操作。这种工具在开发者和系统管理员中特别常见，因为它提供了高效和强大的方式来自动化任务和管理计算机系统。不同的操作系统有不同的命令行解释器，例如Windows上的Command Prompt和PowerShell，Linux和macOS上的Bash shell等。
- Shell 操作系统中为用户提供访问内核所提供之服务的程序
- 命令行解释器是一个单独的软件程序，它可以在用户和操作系统之间提供直接的通讯。非图形命令行解释器用户界面提供运行基于字符的应用程序和实用程序的环境。

以下是一个基于 Rust 语言实现的简单命令行解释器（CLI）的示例，满足在 Linux 环境下实现、支持命令参数解析以及做好错误处理的要求：

### 1. 创建 Rust 项目

在终端中运行以下命令来创建一个新的 Rust 项目：

```
cargo new my_cli
cd my_cli
```

### 2. 编写代码

编辑 `src/main.rs` 文件，实现命令行解释器的基本功能：

```
use std::io::{self, Write};
use std::process::{Command, Stdio};
use std::path::Path;
use std::env;

fn main() {
    // 简单的欢迎信息
    println!("欢迎使用 RustShell v0.1");
    println!("输入 'exit' 退出\n");

    // 主循环
    loop {
        // 显示提示符
        print_prompt();

        // 读取输入
        let input = match read_input() {
```

```
            Ok(input) => input,
            Err(e) => {
                eprintln!("读取输入错误: {}", e);
                continue;
            }
        };

        // 处理命令
        if input.trim() == "exit" {
            break;
        }

        if let Err(e) = execute_command(&input) {
            eprintln!("执行错误: {}", e);
        }
    }

    println!("再见!");
}

/// 打印提示符
fn print_prompt() {
    // 获取当前工作目录
    let current_dir = env::current_dir().unwrap_or_default();

    print!("{}> ", current_dir.display());
    io::stdout().flush().expect("刷新stdout失败");
}

/// 读取用户输入
fn read_input() -> io::Result<String> {
    let mut input = String::new();
    io::stdin().read_line(&mut input)?;
    Ok(input)
}

/// 执行命令
fn execute_command(input: &str) -> io::Result<()> {
    // 分割输入为命令和参数
    let parts: Vec<&str> = input.trim().split_whitespace().collect();
    if parts.is_empty() {
        return Ok(());
    }

    let command = parts[0];
    let args = &parts[1..];

    match command {
        "cd" => {
            // 处理cd命令
            if args.is_empty() {
                // 如果没有参数，切换到HOME目录
                let home = env::var("HOME").unwrap_or_else(|_| "/".to_string());
                env::set_current_dir(Path::new(&home))?;
```

```
            } else {
                // 切换到指定目录
                env::set_current_dir(Path::new(args[0]))?;
            }
            Ok(())
        }
        _ => {
            // 执行其他命令
            let mut child = Command::new(command)
                .args(args)
                .stdin(Stdio::inherit())
                .stdout(Stdio::inherit())
                .stderr(Stdio::inherit())
                .spawn()?;

            child.wait()?;
            Ok(())
        }
    }
}
```

## 功能说明

要求1: Linux环境下实现 代码使用Rust标准库中的进程管理功能，在Linux环境下可以正常运行 实现了基本的shell功能，如命令执行、工作目录显示等

要求2: 支持命令参数解析 通过split_whitespace()方法分割输入字符串，第一个部分作为命令，其余部分作为参数，特殊处理了cd命令，可以改变当前工作目录

要求3: 错误处理 使用Rust的Result类型进行错误处理，对可能的错误情况进行了处理，如: 使用expect和unwrap_or等处理不可恢复错误

## 5. 编译和运行

在项目目录下运行以下命令来编译和运行程序：

```
cargo run
```

然后在命令行解释器中输入各种命令进行测试，例如：

```
ls -l
echo "Hello, World!"
```

如果输入 `exit`，程序将退出。 ###运行结果如下