[Lv4]minishell(1/3)

♡ 5

syamashi
2021年3月20日 06:08

https://github.com/syamashi/minishell

課題準備から。

0. 0) githubリポジトリ作りましょう

共同作業に欠かせません。

・リポジトリ作成、招待、作業までの流れ。

https://qiita.com/future_kame/items/9fa256aea09faa28b357

・作業branchを切って、masterにmargeするまでの操作

```
# 自分のローカルにあるmasterを最新の状態にします
git pull origin master

# 今いるブランチを確認します。
> git branch
* master

# 自分のブランチを作ります。
> git branch syamashi

# ブランチに移動します。
> git checkout syamashi

# (2回目以降) 自分のブランチを最新のmasterにします
> git pull origin master

# --- syamashiブランチで自分の開発をします ---
# masterを汚さないので好きなだけpushして大丈夫

git add *
git commit -m "#104 fix"
git push origin syamashi
```

```
# 修正が完了したら、いよいよmasterにmargeします。
# githubのGUIから、pullrequestを送るのがわかりやすいです。
# Pull requestsタブから、自分の更新が表示されているので、
# pull requestを作成して、margeすればいいです。

# 直接masterに更新する荒業もできます。
git checkout origin master

# syamashiブランチのコードに更新して、pushする
git pull origin syamashi
git add *
git commit -m "syamashi marge"
git push origin master
```

0. 1 githubのissuesタブを活用しましょう。

bugを見つけたら、とにかくissueあげるといいです。

XXX なるissue番号が表示されるので該当issueをfixしたら

```
git commit -m "#20 fix"
```

コメントにissue番号を記載すると、自動でissueと修正commitが連携されます。

さて、早速実装に入ります。

minishellのライフサイクルは、

・標準入力を受け取る

・入力を解析する

・実行結果を返す

を繰り返し、

・"exit"入力、もしくは入力が空のときにctrl+Dで終了

です。

------ 実装 ------

```
#include  <unistd.h>

int main(int ac, char **av, char **envp) {
 execve("/bin/bash", av, envp);
}
```

完

------実装1：入力------

1.1) まず標準入力を受け取ります。

libftフォルダを設置すること、srcフォルダの中にmain.cの記述を想定します。

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

# include "../libft/libft.h"

# define STDERR 2
# define MINISHELL "MINISHELL$ "

void minishell(char **envp);
void sig_int_input();
```

```
void sig_quit_input();

// envpは環境変数
int main(int argc, char **argv, char **envp)
{
   if (argc == 1)
       minishell(envp);
}

void minishell(char **envp)
{
  char *line;
  int  ret;

  ret = 0;
  while(1)
  {
      char *line;

      ft_putstr_fd(MINISHELL, STDERR);
       // ctrl+C, ctrl+\ の指示
       if (signal(SIGINT, sig_int_input) == SIG_ERR)
       {
           ft_putstr_fd(strerror(errno), STDERR);
           exit (1);
       }
       if (signal(SIGQUIT, sig_quit_input) == SIG_ERR)
       {
           ft_putstr_fd(strerror(errno), STDERR);
           exit (1);
       }
      // 文字を受け取る
      if ((get_next_line(0, &line) == 0))
        {
            ft_putstr_fd("exit\n", STDERR);
            exit (ret);
        }
      ft_putstr_fd(line, STDERR);
      ft_putstr_fd("\n", STDERR);
      free(line);
  }
}


void    sig_int_input()
{
    ft_putstr_fd("\b\b  \b\n", STDERR);
    ft_putstr_fd(MINISHELL, STDERR);
}

void    sig_quit_input()
{
    ft_putstr_fd("\b\b  \b\b", STDERR);
```

・int mainの第三引数を指定すると、環境変数が受け取れます。

Q. 環境変数？

あらかじめ設定されている便利変数です。コマンドの実行に必要なUSER名やPATHなどが格納されています。minishellをexitすれば復元されるので、恐れず変更して大丈夫です。

bash上で確認できます。

```
> export | grep USER
declare -x USER="syamashi"

> echo $USER
syamashi
```

・get_next_lineの中のread(buf, 0)で入力待機状態になります。

Q. get_next_lineの戻り値は？

A.

Enterで文末に"\n"が付与。return (1)されます。

ctrl+DでEOFが付与。return (0)されます。

Q. STDERR?

strace(Linux専用コマンドかも）で動作確認をすると、ループ開始の表示文字はfd2=エラー出力とされています。

straceはbashの動作logを出力するコマンド。課題を進める多くの助けや気づきになります。困ったら頻繁に確認します。

```
strace -o trace.txt -f bash -c 'echo aaa'
```

・signalを設定します。

https://linuxjm.osdn.jp/html/LDP_man-pages/man7/signal.7.html

signalは、ctrlとキーによる中断処理を設定できます。mlx_hookのキー設定と同じ。中断処理はコードのどこにいても、最優先で処理されます。

ctrl+Cは SIG_INT に操作関数を指定。

ctrl+\は SIG_QUIT を設定します。

ctrl+Dは…実はシグナル処理しないです。EOFが送られるだけかも。

このあたりはいずれ遭遇するかもしれません。

```
SIGINT   2  Term    キーボードからの割り込み (Interrupt)
SIGQUIT  3  Core    キーボードによる中止 (Quit)
SIGKILL  9  Term    Kill シグナル
SIGSEGV 11  Core    不正なメモリー参照
SIGPIPE 13  Term    パイプ破壊:
  読み手の無いパイプへの書き出し

SIG_DEL そのシグナルに対するデフォルトの操作を行う.
SIG_IGN シグナルを無視する.
```

Q. /b？

A. 流行ってます。

Ctrl+Cを押すと、^Cが画面上に表示されるのを、無理やり表示させないようにしたものです。

\bはカーソルを戻す。' 'はスペースで上書き表示。

read(0)でなく、termcapで入力受付を実装すれば、この気遣いが無用になるかもしれません。

ここまでで、入力と終了ができるハリボテbashができました。

-> 次：minishell(2/3) : コマンド実行の実装