A Simple Linux Shell

2011037064

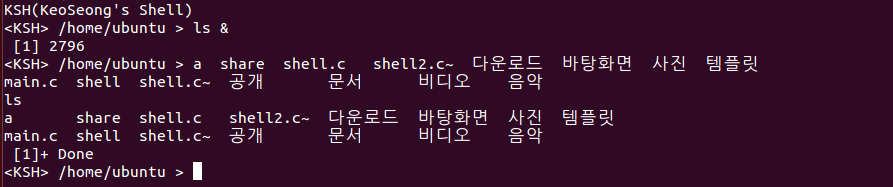
박거성

* In this assignment you will create a simple command shell, called smsh (for Simple Linux Shell), for your Linux system.
* **START !!**
  + gcc –o shell shell.c
  + ./shell

/Users/apple/Desktop/스크린샷 2015-12-07 오전 12.18.06.png

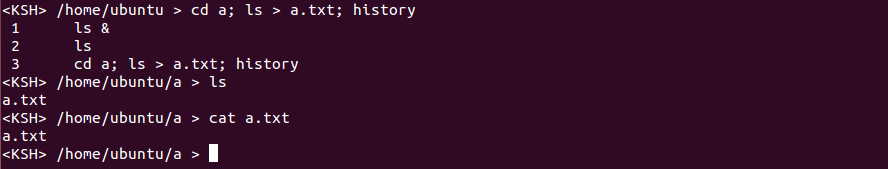
* Compile

1. **foreground and background execution (&)**



* ls 명령어를 background를 실행하고 난 후 다시 ls를 실행

1. **multiple commands separated by semicolons**



* cd a, ls > a.txt, history 3개의 명령어를 순차적으로 실행
* cd a : 디렉토리가 a로 변경
* ls > a.txt : 없는 파일이라 생성되면서 내용에는 ls의 출력물이 들어간 것을 볼 수 있음
* history : 여태까지 실행된 명령어의 현황을 볼 수 있음
* 3개의 명령어가 잘 실행

1. **history command**



* 여태까지 쓴 모든 명령어의 현황이 출력

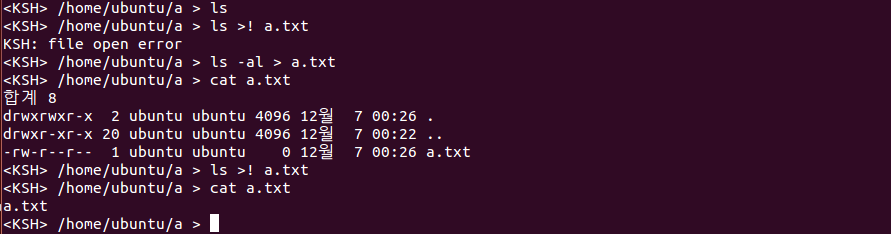


* 추가적으로 history 뒤에 숫자를 입력하면 최근 숫자만큼 history를 띄어주는 것도 구현

1. **shell redirection (>, >>, >!, <)**



* > : 파일이 존재할때 에러 출력



* >! : 파일이 없을 때 에러 출력
* > : 파일 생성, ls –al > a.txt 로 ls –al의 출력을 a.txt로 바꿔서 내용이 들어간 것을 볼 수 있음
* >! : 파일 내용 덮어쓰기가 가능한지 보기 위해서 ls >! a.txt 를 실행했을 때 성공적으로 덮어쓰기가 된 것을 확인할 수 있음

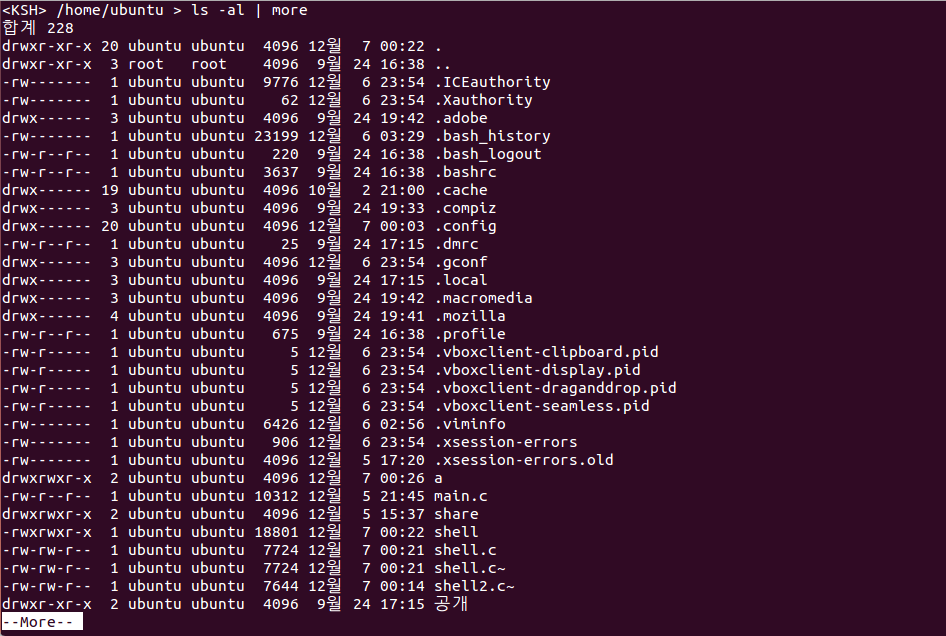


* >> : 파일 내용 이어 붙이기, 그 파일이 존재하지 않을때는 생성하고 존재할때는 Append!



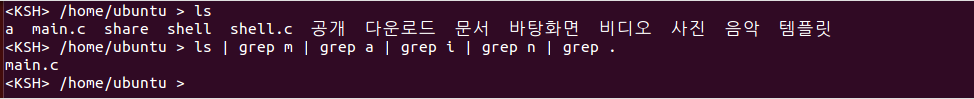
* < : cat의 표준입력을 키보드가 아닌 a.txt의 내용으로 바꾸었으므로 cat < a.txt 명령어 실행 시 정상적으로 a.txt의 명령어가 쉘에 출력

1. **shell pipe (ls –la | more)**



* 싱글 파이프가 정상적으로 실행

1. **Multiple pipe (ls | grep m | grep a | grep i | grep n | grep .)**



* 멀티플 파이프가 정상적으로 실행

1. **cd command**



* Change Directory 가능

***SOURCE CODE***

#include <stdio.h>

#include <unistd.h>

#include <string.h>

#include <stdlib.h>

#include <dirent.h>

#include <ctype.h> // isspace()

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#define MAXCHARS 1024

#define MASHISTORYS 500

#define MAXARGV 128

void print\_prompt(); // print prompt

void commands\_string\_token(char \*command); // tokenize the multiple commands

void change\_directory(char \*command); // change directory

void see\_history(char \*command); // see history

char \*redirection(char \*command); // redirection

void exec\_commands(char \*command); // execute "execvp"

void multiple\_pipe(char \*command); // multiple pipe

char \*trim(char \*s); // remove space

char \*ltrim(char \*s); // remove left space

char \*rtrim(char \*s); // remove right space

int number\_of\_pipe(char \*s); // number of pipe

/\* variables for using history \*/

char history[MASHISTORYS][MAXCHARS];

int command\_count = 0;

/\* variables for using redirection \*/

int redirection\_flag = 0; // 0 : NONE, 1 : '>', 2 : '<', 3 : ">>", 4 : ">!"

char \*redirection\_token;

/\* variable for using background \*/

int amp;

int back\_count = 0;

int main(){

char command[MAXCHARS];

printf("KSH(KeoSeong's Shell)\n");

while(1){

print\_prompt();

fgets(command, MAXCHARS, stdin);

command[strlen(command) - 1] = '\0';

if (command[0] != '\0') {

strcat(history[command\_count], command);

strcat(history[command\_count++], "\n");

commands\_string\_token(command);

}

else continue; // just enter

}

return 0;

}

void print\_prompt(){ // print prompt, <KSH> current working directory >

char \*cwd;

if((cwd = getcwd(NULL, 0)) == NULL){ // get the current working directory

perror("getcwd failed");

exit(1);

}

printf("<KSH> %s > ", cwd);

}

void commands\_string\_token(char \*command){

char \*tmp, \*re\_string;

char \*commands[10]; // number of multiple commands <= 10

int i = 0, j, status;

pid\_t pid;

tmp = strtok(command, ";");

commands[i] = (char \*)malloc(strlen(tmp));

strcpy(commands[i++], tmp);

while (tmp = strtok(NULL, ";")){

commands[i] = (char \*)malloc(strlen(tmp));

strcpy(commands[i++], tmp);

}

for(j = 0; j < i; j++){

redirection\_flag = 0;

/\* processing to identify 4 redirection operators \*/

if (strchr(commands[j], '>') || strchr(commands[j], '<')){

if (strstr(commands[j], ">>") || strstr(commands[j], ">!")){

redirection\_token = (char \*) malloc(2);

if (strstr(commands[j], ">>")) {redirection\_flag = 3; strcpy(redirection\_token, ">>");}

else {redirection\_flag = 4; strcpy(redirection\_token, ">!");}

}

else{

redirection\_token = (char \*) malloc(1);

if (strchr(commands[j], '>')) {redirection\_flag = 1; strcpy(redirection\_token, ">");}

else {redirection\_flag = 2; strcpy(redirection\_token, "<");}

}

}

if (strstr(commands[j], "exit")) exit(0);

if (strchr(commands[j], '&')) {amp = 1; strtok(commands[j], "&"); back\_count++;}

else amp = 0;

change\_directory(commands[j]);

see\_history(commands[j]);

if (!strchr(commands[j], '|')){

if ((pid = fork()) == 0){

if (redirection\_flag != 0) {re\_string = redirection(commands[j]); exec\_commands(re\_string);}

else exec\_commands(commands[j]);

}

if (amp == 0){

waitpid(pid, &status, 0);

if (back\_count != 0) {printf(" [%d]+ Done\n", back\_count); back\_count = 0;}

}

else printf(" [%d] %d\n", back\_count, pid);

}

else multiple\_pipe(commands[j]);

}

}

void change\_directory(char \*command){

char \*cp\_command, \*tmp, \*path;

cp\_command = (char \*)malloc(strlen(command));

strcpy(cp\_command, command);

tmp = strtok(cp\_command, " ");

if (!strcmp(tmp, "cd")) { // change directory

path = strtok(NULL, " ");

if (path == NULL){

path = getenv("HOME");

}

if (chdir(path) < 0){

printf("KSH: cd: %s: No such Directory\n", path);

}

}

}

void see\_history(char \*command){

char \*cp\_command, \*tmp, \*number;

int comp\_num, i;

cp\_command = (char \*)malloc(strlen(command));

strcpy(cp\_command, command);

tmp = strtok(cp\_command, " ");

if (!strcmp(tmp, "history")) { // history

number = strtok(NULL, " ");

if (number == NULL){

for (i = 0; i < command\_count; i++){

printf(" %d\t%s", i + 1, history[i]);

}

}

else{

comp\_num = atoi(number);

if (comp\_num > 0 && comp\_num <= command\_count){

for (i = 0; i < comp\_num; i++){

printf(" %d\t%s", command\_count - comp\_num + i + 1, history[command\_count - comp\_num + i]);

}

}

else {

if (strcmp(number, "0")) printf("KSH: history: %s: invalid value(put the lower number or integer)\n", number);

}

}

}

}

char \*redirection(char \*command){

int fd;

char \*cp\_command, \*left\_command, \*right\_command, \*trim\_left, \*trim\_right, \*re\_argv[MAXARGV];

cp\_command = (char \*)malloc(strlen(command));

strcpy(cp\_command, command);

left\_command = strtok(cp\_command, redirection\_token);

right\_command = strtok(NULL, redirection\_token);

trim\_left = rtrim(left\_command);

trim\_right = trim(right\_command);

if (redirection\_flag == 1){ // ">"

if ((fd = open(trim\_right, O\_CREAT|O\_WRONLY|O\_EXCL, 0644)) < 0) printf("KSH: file already exist\n");

close(1);

dup(fd);

close(fd);

} else if (redirection\_flag == 2){ // "<"

if ((fd = open(trim\_right, O\_RDONLY)) < 0) printf("KSH: file open error\n");

close(0);

dup(fd);

close(fd);

} else if (redirection\_flag == 3){ // ">>"

if((fd = open(trim\_right, O\_CREAT|O\_WRONLY|O\_APPEND, 0644)) < 0) printf("KSH: file open error\n");

close(1);

dup(fd);

close(fd);

} else { // ">!"

if((fd = open(trim\_right, O\_WRONLY|O\_TRUNC, 0644)) < 0) printf("KSH: file open error\n");

close(1);

dup(fd);

close(fd);

}

return trim\_left;

}

void multiple\_pipe(char \*command){

char \*cp\_command, \*tmp;

char \*pipe\_commands[10];

int num, i, j = 0;

int \*pid;

pid = malloc(sizeof(int) \* num);

cp\_command = (char \*)malloc(strlen(command));

strcpy(cp\_command, command);

num = number\_of\_pipe(command);

int pipefds[num][2];

tmp = strtok(cp\_command, "|");

pipe\_commands[j] = (char \*)malloc(strlen(tmp));

strcpy(pipe\_commands[j++], tmp);

while (tmp = strtok(NULL, "|")){

pipe\_commands[j] = (char \*)malloc(strlen(tmp));

strcpy(pipe\_commands[j++], tmp);

}

for (i = 0; i < num; i++){

pipe(pipefds[i]);

fcntl(pipefds[i][0], F\_SETFL, O\_NONBLOCK);

fcntl(pipefds[i][1], F\_SETFL, O\_NONBLOCK);

}

for (i = 0; i < num + 1; i++){

pid[i] = fork();

switch (pid[i]){

case -1:

{

perror("fork error");

}

case 0:

{

if (i == 0){

close(1);

dup(pipefds[i][1]);

close(2);

}

else if (i == num){

close(0);

dup(pipefds[i - 1][0]);

close(2);

}

else {

close(0);

dup(pipefds[i - 1][0]);

close(1);

dup(pipefds[i][1]);

close(2);

}

exec\_commands(pipe\_commands[i]);

}

default:

{

wait(NULL);

}

}

}

}

void exec\_commands(char \*command){

int j, i = 0;

char \*cp\_command, \*tmp, \*argv[MAXARGV];

cp\_command = (char \*)malloc(strlen(command));

strcpy(cp\_command, command);

tmp = strtok(cp\_command, " ");

argv[i] = (char \*)malloc(strlen(tmp));

strcpy(argv[i++], tmp);

while (tmp = strtok(NULL, " ")){

argv[i] = (char \*)malloc(strlen(tmp));

strcpy(argv[i++], tmp);

}

argv[i] = NULL;

execvp(argv[0], argv);

}

char \*rtrim(char \*s) {

char t[MAXCHARS];

char \*end;

strcpy(t, s);

end = t + strlen(t) - 1;

while (end != t && isspace(\*end))

end--;

\*(end + 1) = '\0';

s = t;

return s;

}

char \*ltrim(char \*s) {

char \*begin;

begin = s;

while (\*begin != '\0') {

if (isspace(\*begin))

begin++;

else {

s = begin;

break;

}

}

return s;

}

char \*trim(char \*s) {

return rtrim(ltrim(s));

}

int number\_of\_pipe(char \*s){

int count = 0, i;

for(i = 0; i < strlen(s); i++){

if (\*(s + i) == '|') count++;

}

return count;

}