

Minishell

Lista de comandos implementados:

1. cd
2. pwd
3. clear
4. cp
5. exit
6. help
7. ls
8. man
9. mkdir
10. rmdir
11. rmfile
12. mv
13. cat
14. date

cd

```
int comando_cd(char **arg) {  
    if(arg[1] == NULL) {  
        printf("Error: Cantidad insuficiente de argumentos \n Ver man help");  
    } else {  
        if(chdir(arg[1]) != 0) {  
            printf("Error: Fallo el comando 'cd'\n");  
        }  
    }  
    return 0;  
}
```

pwd

```

int comando_pwd(char **arg) {
    if(arg[1] != NULL) {
        printf("Error: El comando 'pwd' no acepta ningun argumento.\n");
        return 0;
    }
    char cwd[1024];
    chdir("/path/to/change/directory/to");
    getcwd(cwd, sizeof(cwd));
    printf("%s\n", cwd);
    return 0;
}

```

```

miniShell> pwd
/home/miguel/Repository/pucmm/Semestre10/S02/Tareas/7
miniShell> ls
.main.c.swp      .minishell.md.swp  README.md      a.out      main.c
minishell.md
miniShell> cd ..
miniShell> ls
1      2      3      4      6      7
miniShell> pwd
/home/miguel/Repository/pucmm/Semestre10/S02/Tareas

```

cp

```

int comando_cp(char **arg) {
    /* Uso:  cp [nombre_archivo] [/destino/nombre_archivo_copia] */

    FILE *fuente,*destino;
    char c;

    if(arg[2]== NULL) { //cantidad argumentos < 3
        printf("Error: Cantidad insuficiente de argumentos\n");
        return -1;
    } else if(arg[1] == NULL) {
        printf("Error: Cantidad de argumentos\n cp [fuente] [destino]");
    }
    fuente=fopen(arg[1],"r"); //Se abre el archivo fuente para lectura
    destino=fopen(arg[2],"w"); //Se abre el archivo destino para escritura
    if(fuente==NULL || destino==NULL) {
        printf("Error: Fallo en el archivo fuente o destino\n");
        return 0;
    }
    while((c=fgetc(fuente))!=EOF) { //Se lee el archivo fuente caracter a
caracter
        fputc(c,destino); //Se escribe en el archivo destino
    }
    fclose(fuente);
    fclose(destino); //Se cierran ambos archivos

    return 0;
}

```

```
miniShell> ls
README.md      a.out      img      main.c      minishell.md  new_folder
miniShell> touch prueba.txt
miniShell> cp prueba.txt ../prueba.txt
miniShell> cd ..
miniShell> ls
0      1      2      3      4      6      7      8      mshell  prueba.txt
```

mv

```
int comando_mv(char **arg) {
    if(comando_cp(arg) == 0) {
        printf("%s\n", arg[1]);
        remove(arg[1]);
    }
    return 0;
}
```

```
miniShell> mv prueba.txt bien.txt
prueba.txt
miniShell> ls
0      1      2      3      4      6      7      8      bien.txt  mshell
```

touch

```
int comando_touch(char **arg) {
    if (arg[1] == NULL) {
        printf("Error: Cantidad insuficiente de argumentos\n");
    } else {
        FILE* out = fopen(arg[1], "w");
        fclose(out);
    }
    return 0;
}
```

```
miniShell> ls
README.md      a.out      img      main.c      minishell.md  new_folder
miniShell> touch prueba.txt
miniShell> ls
README.md      a.out      img      main.c      minishell.md  new_folder  prueba.txt
```

rmfile

```
int comando_rmfile(char **arg) {
    if (arg[1]==NULL) {
        printf("Error: El comando 'rmfile' debe tener un argumento\n");
    } else if (arg[2]!=NULL) {
        printf("Error: El comando 'rmfile' no puede tener mas de un argumento\n");
    }

    if(remove(arg[1]) == -1)
        perror("Error: No se puedo eliminar el archivo");
    return 0;
}
```

```
miniShell> ls
README.md      a.out      img      main.c      minishell.md  new_folder
miniShell> touch prueba.txt
miniShell> ls
README.md      a.out      img      main.c      minishell.md  new_folder  prueba.txt
miniShell> rmfile prueba.txt
miniShell> ls
README.md      a.out      img      main.c      minishell.md  new_folder
```

cat

```
int comando_cat(char **arg) {
    if(arg[2] != NULL) {
        printf("Solo un argumento");
        return 0;
    }

    if(arg[1] == NULL) {
        printf("Se necesita un argumento");
        return 0;
    }

    FILE* fuente;
    fuente = fopen(arg[1], "r");

    if(fuente == NULL) {
        printf("Archivo: %s no existe\n", arg[1]);
        return 0;
    }
    char c;

    while((c=fgetc(fuente))!=EOF) { //Se lee el archivo fuente caracter a
caracter
        printf("%c",c);
    }
    fclose(fuente);
    return 0;
}
```

```
miniShell> cat README.md
# A Construir un Mini-Shell para Linux

## 1.- Introducción

Esta practica se diseña para proporcionar los conocimientos y la comprensión de cómo crear y utilizar procesos en el sistema operativo Linux. Usted también aprenderá cómo encontrar, leer, y utilizar documentación técnica a través de páginas de manual.

Dado que muchos sistemas operativos han tomado prestado muchas ideas de LINUX en el conocimiento que se adquiere en esta práctica nos sirve para comprender mejor otros sistemas.

Preparación:
Antes de comenzar el laboratorio debes examinar los ejemplos que se señalan, y resolver preguntas se proporcionan con los ejemplos. Tenga en cuenta que es importante estudiar las páginas de los manuales utilizando el comando man.

1.1 La tarea
La tarea consiste en escribir un programa que actúa como un intérprete de comandos simple (shell) para Linux. El programa permitirá al usuario introducir comandos hasta que
```

date

```
int comando_date(char **arg) {
    if(arg[1] != NULL) {
        perror("No acepta ningun argumento");
        return 0;
    }
    time_t t = time(NULL);
    struct tm tm = *localtime(&t);
    printf("%d-%02d-%02d %02d:%02d:%02d\n", tm.tm_year + 1900, tm.tm_mon + 1,
tm.tm_mday, tm.tm_hour, tm.tm_min, tm.tm_sec);
    return 0;
}
```

```
miniShell> date
2020-11-29 15:20:51
```

mkdir

```
int comando_mkdir(char **arg) {
    if (arg[2] != NULL) {
        printf("Error: El comando 'mkdir' solo acepta un argumento\n");
        return 0;
    }
    char tmp[256];
    char *p = NULL;
    size_t len;

    snprintf(tmp, sizeof(tmp), "%s", arg[1]);
    len = strlen(tmp);
    if(tmp[len - 1] == '/') {
        tmp[len - 1] = 0;
    }
    for(p = tmp + 1; *p; p++) {
        if(*p == '/') {
```

```

        *p = 0;
        mkdir(tmp, S_IRWXU);
        *p = '/';
    }
}

mkdir(tmp, S_IRWXU);

return 0;
}

```

```

miniShell> ls
README.md      a.out      img      main.c      minishell.md
miniShell> mkdir new_folder
miniShell> ls
README.md      a.out      img      main.c      minishell.md      new_folder

```

rmdir

```

int comando_rmdir(char **arg) {
    char* dir = arg[1];
    int ret = 0;
    FTS *ftsp = NULL;
    FTSENT *curr;

    if (arg[1]==NULL) {
        printf("Error: El comando 'rmdir' debe tener un argumento\n");
    } else if (arg[2]!=NULL) {
        printf("Error: El comando 'rmdir' no puede tener mas de un
argumento\n");
    }

    /* Casting necesario en C porque fts_open() toma un "char * const *", en vez
de "const char * const *", el
cual solamente esta permitido en C++. La llamada a fts_open() no modifica al
argumento */
    char *files[] = { (char *) dir, NULL };

    // FTS_NOCHDIR - Evitar cambiar el cwd, lo que podria causar un
comportamiento inesperado en multithreading
    // FTS_PHYSICAL - Previene eliminar archivos por fuera del directorio
especificado
    // FTS_XDEV - No cruzar los limites del sistema de archivos
    ftsp = fts_open(files, FTS_NOCHDIR | FTS_PHYSICAL | FTS_XDEV, NULL);
    if (!ftsp) {
        fprintf(stderr, "%s: Fallo en fts_open: %s\n", dir, strerror(errno));
        ret = -1;
        goto finish;
    }

    while ((curr = fts_read(ftsp))) {
        switch (curr->fts_info) {
            case FTS_NS:
            case FTS_DNR:
            case FTS_ERR:
                fprintf(stderr, "%s: Fallo en fts_read: %s\n",

```

```

curr->fts_accpath, strerror(curr->fts_errno));

    break;

    case FTS_DC:
    case FTS_DOT:
    case FTS_NSOK:
        //Codigo no alcanzado a menos que FTS_LOGICAL, FTS_SEEDOT, o
        FTS_NOSTAT hayan sido
        //pasados a fts_open()
        break;

    case FTS_D:
        // No hacer nada. Se necesita busqueda depth-first, entonces los
        directorios son
        // eliminados en FTS_DP
        break;

    case FTS_DP:
    case FTS_F:
    case FTS_SL:
    case FTS_SLNONE:
    case FTS_DEFAULT:
        if (remove(curr->fts_accpath) < 0) {
            fprintf(stderr, "%s: Error: Fallo el comando 'rmdir': %s\n",
                    curr->fts_path, strerror(errno));
            ret = -1;
        }
        break;
    }
}

finish:
    if (ftsp) {
        fts_close(ftsp);
    }

    return ret;
}

```

```

miniShell> ls
README.md      a.out      img      main.c      minishell.md      new_folder      prueba.txt

miniShell> cd new_folder
miniShell> ls

miniShell> touch archivo.txt
miniShell> ls
archivo.txt
miniShell> cd ..
miniShell> ls
README.md      a.out      img      main.c      minishell.md      new_folder      prueba.txt

miniShell> rmdir new_folder
miniShell> ls
README.md      a.out      img      main.c      minishell.md      prueba.txt

```

man

```
miniShell> man
Puede buscar informacion sobre los comandos escribiendo:
  man [nombre_comando]
Ejemplo:    man ls

miniShell> man ls
Comando 'ls':
  Descripcion: Listar el contenido de un directorio.
  Uso: ls
  Limitaciones: Formato de salida (agrupamiento, orden alfabetico,etc)
```

echo y exit

```
miniShell> echo Hola mundo
Hola mundo
miniShell> exit
Bye
```