`access` (man 2 access)

`int access (const char *pathname, int mode);`
Here, the first argument takes the path to the *directory/file* and the second argument takes flags *R_OK, W_OK, X_OK or F_OK*.

- **F_OK flag** : Used to check for the existence of file.
- **R_OK flag** : Used to check for read permission bit.
- **W_OK flag** : Used to check for write permission bit.
- **X_OK flag** : Used to check for execute permission bit.

Note: If access () cannot access the file, it will return -1 or else it will be 0.
`Link:` https://www.geeksforgeeks.org/access-command-in-linux-with-examples/

```c
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

extern int errno;

int main(int argc, const char *argv[]) {
    int fd = access("sample.txt", F_OK);
    if (fd == -1) {
        printf("Error Number: %d\n", errno);
        perror("Error Description:");
    } else {
        printf("No error\n");
    }
    return 0;
}
```

```c
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

extern int errno;

int main(int argc, const char *argv[]) {
    int fd = access("sample.txt", (R_OK | W_OK) & X_OK);
    if (fd == -1) {
        printf("Error Number: %d\n", errno);
        perror("Error Description:");
    } else {
        printf("No error\n");
```

```
    }
    return 0;
}
```

```
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

extern int errno;

int main(int argc, const char *argv[]) {
    int fd = access("sample.txt", R_OK & W_OK & X_OK);
    if (fd == -1) {
        printf("Error Number: %d\n", errno);
        perror("Error Description:");
    } else {
        printf("No error\n");
    }
    return 0;
}
```

`chdir` (man 2 chdir)

```
int chdir(const char *path);
```

The **chdir** command is a system function (system call) that is used to change the current working directory. On some systems, this command is used as an alias for the shell command cd. chdir changes the current working directory of the calling process to the directory specified in path.

Parameter: Here, the *path* is the Directory path that the user want to make the current working directory.
Return Value: This command returns zero (0) on success. -1 is returned on an error and errno is set appropriately.
Note: It is declared in unistd.h.

Link: https://www.geeksforgeeks.org/chdir-in-c-language-with-examples/?ref=gcse

```c
#include<stdio.h>

// chdir function is declared
// inside this header
#include<unistd.h>
int main()
{
    char s[100];

    // printing current working directory
    printf("%s\n", getcwd(s, 100));

    // using the command
    chdir("..");

    // printing current working directory
    printf("%s\n", getcwd(s, 100));

    // after chdir is executed
    return 0;
}
```

`close` (man 2 close)

3. C close

The close() function in C tells the operating system that you are done with a file descriptor and closes the file pointed by the file descriptor. It is defined inside **<unistd.h>** header file.

Syntax of close() in C

```
int close(int fd);
```

Parameter

- **fd:** File descriptor of the file that you want to close.

Return Value

- **0** on success.
- **-1** on error.

Link: https://www.geeksforgeeks.org/input-output-system-calls-c-create-open-close-read-write/?ref=gcse

```c
// C program to illustrate close system Call
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
	int fd1 = open("foo.txt", O_RDONLY);
	if (fd1 < 0) {
		perror("c1");
		exit(1);
	}
	printf("opened the fd = % d\n", fd1);

	// Using close system Call
	if (close(fd1) < 0) {
		perror("c1");
		exit(1);
	}
	printf("closed the fd.\n");
}
```

`closedir` (man 3 closedir)

`execve` (man 2 execve)

`exit` (man 3 exit)

`_exit` (man 2 _exit)

`fflush` (man 3 fflush)

fflush() is typically used for output stream only. Its purpose is to clear (or flush) the output buffer and move the buffered data to console (in case of stdout) or disk (in case of file output stream). Below is its syntax

- `fflush(stdin);`

`fork` (man 2 fork)

`free` (man 3 free)

//////////////////

`getcwd` (man 3 getcwd)

printf("%s\n", getcwd(s, 100));

`getline` (man 3 getline)

`getpid` (man 2 getpid)

`isatty` (man 3 isatty)

`kill` (man 2 kill)

`malloc` (man 3 malloc)

//////////

`open` (man 2 open)

C open

The open() function in C is used to open the file for reading, writing, or both. It is also capable of creating the file if it does not exist. It is defined inside **<unistd.h>** header file and the flags that are passed as arguments are defined inside **<fcntl.h>** header file.

Syntax of open() in C

```
int open (const char* Path, int flags);
```

Link: https://www.geeksforgeeks.org/input-output-system-calls-c-create-open-close-read-write/?ref=gcse

`opendir` (man 3 opendir)

`perror` (man 3 perror)

`read` (man 2 read)

C read

From the file indicated by the file descriptor fd, the read() function reads the specified amount of bytes **cnt** of input into the memory area indicated by **buf**. A successful read() updates the access time for the file. The read() function is also defined inside the <unistd.h> header file.

Syntax of read() in C

```
size_t read (int fd, void* buf, size_t cnt);
```

Parameters

- **fd:** file descriptor of the file from which data is to be read.
- **buf:** buffer to read data from
- **cnt:** length of the buffer

Return Value

- return Number of bytes read on success
- return 0 on reaching the end of file
- return -1 on error
- return -1 on signal interrupt

`readdir` (man 3 readdir)

`signal` (man 2 signal)

`stat` (__xstat) (man 2 stat)

`lstat` (__lxstat) (man 2 lstat)

`fstat` (__fxstat) (man 2 fstat)

`strtok` (man 3 strtok)

`wait` (man 2 wait)

`waitpid` (man 2 waitpid)

`wait3` (man 2 wait3)

`wait4` (man 2 wait4)

`write` (man 2 write)

C write

Writes cnt bytes from buf to the file or socket associated with fd. cnt should not be greater than INT_MAX (defined in the limits.h header file). If cnt is zero, write() simply returns 0 without attempting any other action.

The write() is also defined inside **<unistd.h>** header file.

Syntax of write() in C

```
size_t write (int fd, void* buf, size_t cnt);
```

Parameters

- **fd:** file descriptor
- **buf:** buffer to write data to.
- **cnt:** length of the buffer.

Return Value

- returns the number of bytes written on success.
- return 0 on reaching the End of File.
- return -1 on error.
- return -1 on signal interrupts.