```
int open (const char *pathname, int flags, mode
          -t mode);
```

* The pathname argument is simply a string with the full or relative pathname to the file to be opened.
* The third parameter specifies the UNIX file mode to be used when creating a file and should be present if a file may be created.
* The second parameter, flags, is one of O_RDONLY, O_WRONLY, or O_RDWR, optionally OR-ed with additional flag.

FLAGS FOR THE open() CALL

flag	Description
O_RDONLY	open file for read-only access
O_WRONGLY	open file for write-only access
O_RDWR	open file for read and write access
O_CREAT	Create the file if it does not exist
O_EXCL	fail if the file already exists
O_NOCTTY	Don't become controlling tty if opening tty and the process had no controlling tty.
O_TRUNC	Truncate the file to length 0 if it exists

int open (const char *pathname, int flags, mode_t mode);

* The pathname argument is simply a string with the full or relative pathname to the file to be opened.
* The third parameter specifies the UNIX file mode to be used when creating a file and should be present if a file may be created.
* The second parameter, flags, is one of O_RDONLY, O_WRONLY, or O_RDWR, optionally OR-ed with additional flags.

FLAGS FOR THE open() CALL

flag	Description
O_RDONLY	open file for read-only access
O_WRONGLY	open file for write-only access
O_RDWR	open file for read and write access
O_CREAT	create the file if it does not exist
O_EXCL	fail if the file already exists
O_NOCTTY	Don't become controlling tty if opening tty and the process had no controlling tty.
O_TRUNC	Truncate the file to length 0 if it exists

I/O routine

* file descriptor — based on I/O
* A file descriptor is simply an integer that is used an index into a table of open files associated with each process
* The values 0, 1 and 2 are special and refer to the stdin, stdout and stderr streams.
* These three streams normally connect to the user's terminal but can be redirected.
* There are many security implications to using file descriptor I/O and file pointer I/O

Calls that use File Descriptor

* A no. of system calls use file descriptors.
* Calls, including the function prototypes from the man pages and/or header files.
* Most of these calls return a value of -1 in the event of error and set the variable errno to error code.
* Error codes are documented in the man pages for the individual system calls and in the man page for errno
* The perror() function can be used to print an error message based on error code.
* Some call are vulnerable, other are used to fix vulnerabilities.
* The calls that take file descriptors are much safer than those that take filenames.

The open() Call - * The open() call is used to open a file.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open (const char *pathname, int flags),
```

Variable	Description
ASFLAGS	flags for the assembler program no default
CFLAGS	flags for the C compiler; no default
LDFLAGS	flags for the linker (ld); no default

---

implicit rules → * In addition to the rules that you explicitly specify in a makefile, which are called explicit rules,

* make comes with a comprehensive set of implicit, or predefined rules.

* implicit rules simplify makefile maintenance.

pattern rules → * pattern rules provide a way around the limitations of make's implicit rules by defining your own implicit rules.

* pattern rules look like normal rules, except that the target contains exactly one character (%) that matches any nonempty string.

* The dependencies of such a rule also use % in order to match the target.

* example - The rule %.o : %.c

Tells make to build any object file a.o from a source file b.c

Automatic variables

variable	Description
$@	The filename of a rule's target
$<	The name of the 1st dependency in a rule.
$^	space-delimited list of all the dependencies in a rule
$?	space-delimited list of all the dependencies in a rule that are newer than the target
$(@D)	The directory part of a target filename, if the target is in a subdirectory.
$(@F)	The filename part of a target filename, if the target is in a subdirectory.

Predefined variables for program Names and flags.

variable	Description
AR	Archive-maintenance program; default value = ar
CC	program for compiling C programs; default value = cc
AS	program to do assembly; default value = as
CPP	preprocessor program; default value = $(CC) -E
RM	program to remove file; default value = "rm -f"
ARFLAGS	flags for the archive-maintenance program default = rv

* OBJS and HDRS expand to their value each time they are referenced.
* make uses two kinds of variables - recursively-expanded and simply expanded.
* Recursively-expanded variables are expanded variables as they are referenced; if the expansion contains another variable reference, it is also expanded.
* The expansion continues until no further variables exist to expand hence the name, "recursively-expanded!!

```
TOPDIR = /home/kwall/myproject
SRCDIR = $(TOPDIR)/src
```

↳ Thus SRCDIR will have the value /home/kwall/myproject/src

```
CC = gcc
CC = $(CC)-o
```

↳ `CC = gcc-o`

Environment, Automatic, and predefined Variable

* environment variable → When it starts, make reads every variable defined in its environment and creates variables with the same name and value.
* make provides a long list of predefined and automatic variables, I

variable → * To simplify editing and maintaining makefile; we use the variable.

* A variable is simply a name defined in a make file that represents a string of text: this text is called the variable's value.

```
VARNAME = some_text [...]
```

* To obtain VARNAME's value, enclose it in parenthesis and prefix it with a $.

```
$(VARNAME)
```

* Variables are usually defined at the top of a make-file.

* By convention, make-file variables are all uppercase.

* If the value changes, you only need to make one change instead of many, simplifying makefile maintenance.

```
1. OBJS = editor.o screen.o keyboard.o
2 HDRS = editor.h screen.h keyboard.h
3 editor : $(OBJS)
4 gcc -o editor $(OBJS)
5
6 editor.o : editor.c $(HDRS)
7 gcc -c editor.c
8
9 screen.o : screen.c screen.h
10 gcc -c screen.c
11
12 keyboard.o : keyboard.c keyboard.h
13 gcc -c keyboard.c
14
15 .PHONY : clean
16
17 clean
18 rm editor $(OBJS)
```