

Essentially a **makefile** contains a set of rules used to build an application. The first rule seen by make is used as the default rule. A rule consists of three parts: the *target*, its *prerequisites*, and the *command(s)* to perform:

```
target: prereq1 prereq2  
        commands
```

- The *target* is the file or thing that must be made.
- The prerequisites or dependents are those files that must exist before the target can be successfully created.
- *commands* are those shell commands that will create the target from the prerequisites.

```
foo.o: foo.c foo.h  
      gcc -c foo.c
```

- *target*: foo.o
- prerequisites: foo.c, foo.h
- commands: gcc -c foo.c

Note: the commands should be indented in makefile using spaces only

My Example

```

C++ first.cpp > main
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4
5  int main()
6  {
7      int inp=25;
8      cout << "Sqrt of " << inp << " is " << sqrt(inp) << endl;
9      return 0;
10 }
11
12

```

What I need is, I want to create a output binary and run the binary to see the result. The makefile in this case can be:

```

Makefile
1  all: getout print
2
3  getout: first.cpp
4      g++ first.cpp -o output
5
6  print: getout
7      ./output

```


Since rules are so important in make , there are a number of different kinds of rules.

- Explicit rules (Discussed above and mostly used)
- Pattern rules (Uses bash-like globbing rules instead of defining filenames)
- Implicit rules (a database of rules)

There are static pattern rules and suffix rules as well.

What is phony target?

Targets that do not represent files are known as phony targets. e.g., clean target below

 Makefile
1 1 all: getout print
0 2
3 getout: first.cpp
4 g++ first.cpp -o output
5
6 print: getout
7 ./output
8
9 clean:
10 rm output
11
12 .PHONY: clean

Now make will always execute the commands associated with clean even if a file named clean exists. Phony targets are always out of date, so they always execute and they always cause their dependent (the target associated with the prerequisite) to be remade.

Automatic variables

There are six “core” automatic variables:

- `$@` The filename representing the target.
- `%` The filename element of an archive member specification.
- `$<` The filename of the first prerequisite.
- `$?` The names of all prerequisites that are newer than the target, separated by spaces.
- `^` The filenames of all the prerequisites, separated by spaces. This list has duplicate filenames removed since for most uses, such as compiling, copying, etc., duplicates are not wanted.
- `+` Similar to `^`, this is the names of all the prerequisites separated by spaces, except that `+` includes duplicates. This variable was created for specific situations such as arguments to linkers where duplicate values have meaning.
- `*` The stem of the target filename. A stem is typically a filename without its suffix. (We’ll discuss how stems are computed later in the section “Pattern Rules.”) Its use outside of pattern rules is discouraged.

The above makefile modified by using the automatic variables is as follows:

```
Makefile
12 1  .PHONY: all
11 2  all: output print
10 3
9   4  output: first.cpp
8   5      g++ ^ -o $@
7   6
6   7  print: output
5   8      ./$<
4   9
3  10  .PHONY: clean
2  11  clean:
1  12      rm output
13
```

