

There are two ways to set variables using bash shell.

[1] Variables that do not get passed on to children.

Using the following command a variable called \$NAME is created and it has the value “MyLaptop”.

```
NAME=MyLaptop
```

Check this out by running that and using the following command to print out the variable at the command prompt.

```
echo $NAME
```

Variables set like this are not passed on to the child shells or programs that are launched from the current shell. See the following screenshot for example. The variable \$NAME is available only in the parent shell and not in the child shell.



```
gphani@gphanilaptop: ~  
File Edit View Search Terminal Help  
gphani@gphanilaptop:~$ NAME=MyLaptop  
gphani@gphanilaptop:~$ echo $NAME  
MyLaptop  
gphani@gphanilaptop:~$ bash  
gphani@gphanilaptop:~$ echo $NAME  
  
gphani@gphanilaptop:~$ exit  
exit  
gphani@gphanilaptop:~$ echo $NAME  
MyLaptop  
gphani@gphanilaptop:~$ █
```

We are entering a child shell

This is run inside child shell

We are exiting the child shell

We are back to parent shell

[2] Variables that do get passed on to children.

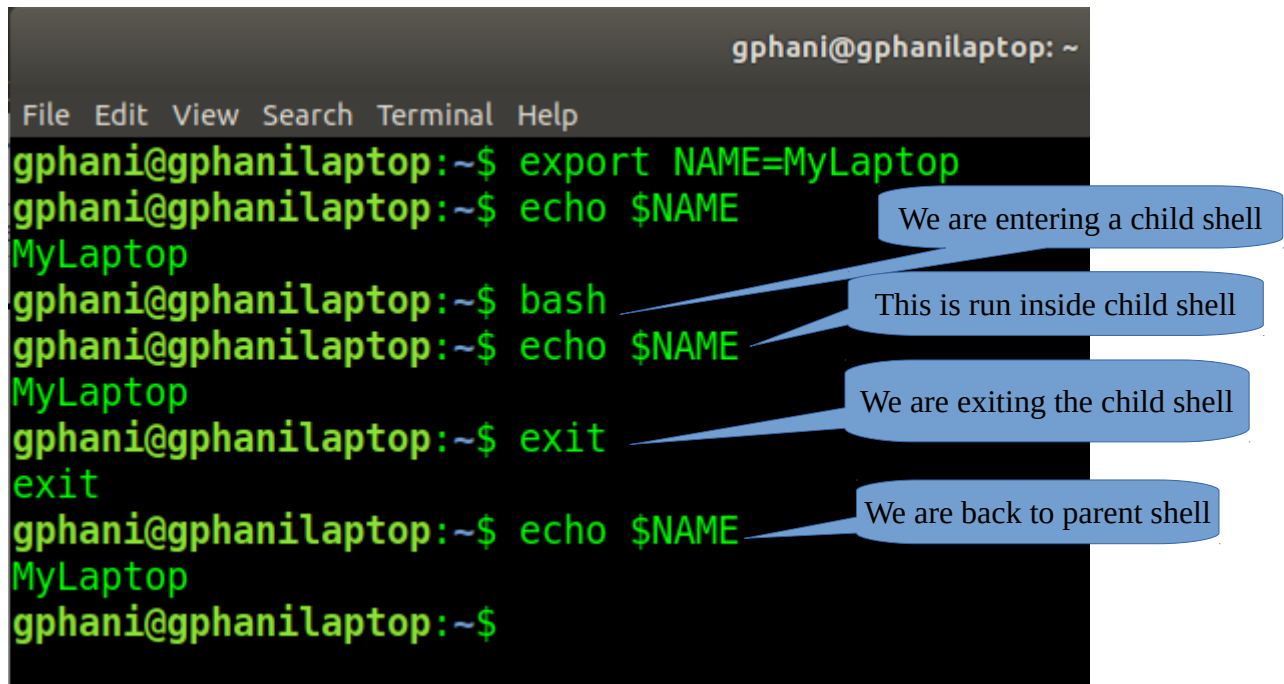
Using the following command a variable called \$NAME is created and it has the value “MyLaptop”.

```
export NAME=MyLaptop
```

Check this out by running that and using the following command to print out the variable at the command prompt.

```
echo $NAME
```

Variables set like this are passed on to the child shells or programs that are launched from the current shell. See the following screenshot for example. The variable \$NAME is available not only in the parent shell but also in the child shell.



The screenshot shows a terminal window with a menu bar (File, Edit, View, Search, Terminal, Help) and a title bar (gphani@gphanilaptop: ~). The terminal output is as follows:

```
gphani@gphanilaptop:~$ export NAME=MyLaptop
gphani@gphanilaptop:~$ echo $NAME
MyLaptop
gphani@gphanilaptop:~$ bash
gphani@gphanilaptop:~$ echo $NAME
MyLaptop
gphani@gphanilaptop:~$ exit
exit
gphani@gphanilaptop:~$ echo $NAME
MyLaptop
gphani@gphanilaptop:~$
```

Four blue callout boxes provide context for the terminal actions:

- "We are entering a child shell" points to the `bash` command.
- "This is run inside child shell" points to the `echo $NAME` command inside the child shell.
- "We are exiting the child shell" points to the `exit` command.
- "We are back to parent shell" points to the `echo $NAME` command after exiting the child shell.

Application to make utility.

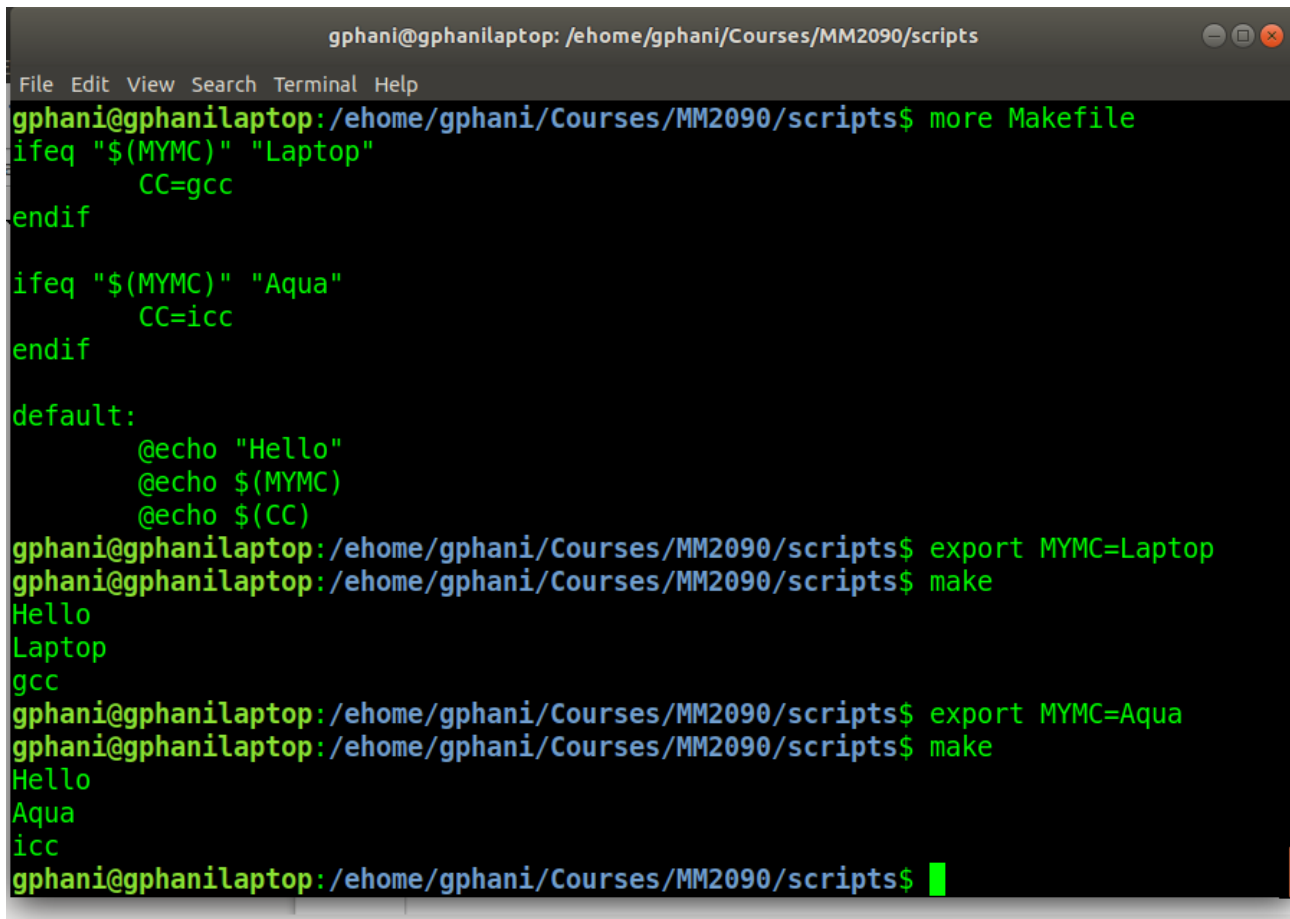
Using shell variables like above, one can control the behavior of the recipe that can be used in Makefile. All shell variables set using export are available within Makefile like other aliases or macros created within. These can be used to control the choice of compiler depending on the hostname or the version / type of OS etc.,

The following content of Makefile illustrates the conditional compilation as follows. When the environment variable MYMC is set to "Laptop" then the gcc compiler is used. This is achieved by setting the variable CC to the value gcc.

When the environment variable MYMC is set to "Aqua" then the icc compiler is used. This is achieved by setting the variable CC to the value icc.

The ifeq loop allows for this conditional setting of variable CC for the purpose.

One can expand this feature to create one master Makefile that would work on multiple hosts and environments. This is particularly useful when one uses ones laptop for code development and then moves the code to a remote computer for production run. If the laptop and the remote computer have different compiler tools or operating system then a master Makefile helps keep the code repository including the Makefile the same all through.

A terminal window titled 'gphani@gphanilaptop: /ehome/gphani/Courses/MM2090/scripts' showing the execution of a Makefile. The Makefile contains conditional compilation rules for 'Laptop' (using gcc) and 'Aqua' (using icc). The user runs 'make' twice, first with MYMC=Laptop and then with MYMC=Aqua. The output shows the compiler being used and the word 'Hello' being printed.

```
gphani@gphanilaptop: /ehome/gphani/Courses/MM2090/scripts$ more Makefile
ifeq "$(MYMC)" "Laptop"
    CC=gcc
endif

ifeq "$(MYMC)" "Aqua"
    CC=icc
endif

default:
    @echo "Hello"
    @echo $(MYMC)
    @echo $(CC)
gphani@gphanilaptop: /ehome/gphani/Courses/MM2090/scripts$ export MYMC=Laptop
gphani@gphanilaptop: /ehome/gphani/Courses/MM2090/scripts$ make
Hello
Laptop
gcc
gphani@gphanilaptop: /ehome/gphani/Courses/MM2090/scripts$ export MYMC=Aqua
gphani@gphanilaptop: /ehome/gphani/Courses/MM2090/scripts$ make
Hello
Aqua
icc
gphani@gphanilaptop: /ehome/gphani/Courses/MM2090/scripts$
```

Exported variables such as CPATH, C_INCLUDE_PATH, CPLUS_INCLUDE_PATH, OBJC_INCLUDE_PATH, LIBRARY_PATH, LIBPATH, LDPATH etc., are important for compilation. Ask and find out what variables are being used for compilation when you download a code from elsewhere. The locations to be searched for the header file, static libraries for linking at compilation stage and dynamic libraries for linking at execution stage need to be informed to the utility that is being used by the user. Often a mistake in such path variables leads to the codes not being compiled. Watch out for all the exported variables and their values and ensure they are set similarly in your system before embarking on compilation of a code downloaded from elsewhere.

Homework:

- [1] Prepare a Makefile that performs conditional compilation depending on the [architecture](#) of the machine. Ensure that this information is passed on to the compiler options explicitly. Use any of your old codes for this example.
- [2] Prepare a Makefile that first checks for availability of the compiler before performing the compilation. In case the compiler is not installed, the make utility should display a helpful message while exiting. Try this on any of your old codes.