GNU Debugger Tutorial

GDB, short for GNU Debugger, is the most popular debugger for UNIX systems to debug C programs. A debugger is a program that runs other programs, allowing the user to exercise control over these programs, and to examine variables when problems arise.

GNU Debugger helps you to get information about the following:

- If a core dump happened, then what statement or expression did the program crash on?
- If an error occurs while executing a function, what line of the program contains the call to that function, and what are the parameters?
- What are the values of program variables at a particular point during execution of the program?
- What is the result of a particular expression in a program?

Note: - GDB cannot be used for programs that compile with errors and it does not help in fixing those errors.

GDB - Installation

Before you go for installation, check if you already have gdb installed on your Unix system by issuing the following command:

```
$gdb –help
```

If GDB is installed, then it will display all the available options within your GDB. If GDB is not installed, then proceed for a fresh installation.

You can install GDB on your system by following the simple steps discussed below.

Step 1: Make sure you have the prerequisites for installing gdb:

- An ANSI-compliant C compiler (gcc is recommended - note that gdb can debug codes generated by other compilers)
- 115 MB of free disk space is required on the partition on which you're going to build gdb.
- 20 MB of free disk space is required on the partition on which you're going to install gdb.

Step 2: Use the following command to install gdb on linux machine.

```
$ sudo apt-get install libc6-dbg gdb valgrind
```

Step 3: Now use the following command to find the help information.

```
$gdb –help
```

You now have gdb installed on your system and it is ready to use

GDB Debugging steps

Step 1: To let GDB be able to read all that information line by line from the symbol table, we need to compile it a bit differently. Normally we compile our programs as:

```
$ gcc sample.c  -o  sample
```

Instead of doing this, we need to compile with the -g flag as shown below:

```
$ gcc –g sample.c  -o  sample
```

Step 2: You need to select the below command to run the executable file

```
$ gdb a.out
```

Opens GDB with file a.out, but does not run the program. You'll see a prompt (gdb) - all examples are from this prompt.

Step 3: If you want to check line by line execution of a function, then you need to set breakpoint. U can specify breakpoint to more than one function also.

```
(gdb) b main
```

In the above command, breakpoint is set to function main().

Step 4: To run a program, we have to use command "r" and if you want to pass input through command line you need to pass input at this stage.

If you don't want to pass input in command line then

```
(gdb) r
```

If you want to pass input in command line then you have to pass as shown below.

```
(gdb) r arg1 arg2 arg3
```

Step 5: Since we set breakpoint to function, step by step execution of function will be carried. If we want to move from current instruction to next, then we have to use "n".

(gdb) n

If you want to print value of variable then we have to use “p <var_name>”.

(gdb) p num

The above command prints the value of variable num. We have to repeat this step until control come out of the function.

Step 6: Once you finish with execution, if we want to come out of gdb program we have to use “q” to quit the program.

(gdb) q

Do you want to continue(y/n): y

$

GDB offers a big list of commands, however the following commands are the ones used most frequently:

b main - Puts a breakpoint at the beginning of the program

b - Puts a breakpoint at the current line

b N - Puts a breakpoint at line N

b +N - Puts a breakpoint N lines down from the current line

b fn - Puts a breakpoint at the beginning of function "fn"

d N - Deletes breakpoint number N

info break - list breakpoints

r - Runs the program until a breakpoint or error

c - Continues running the program until the next breakpoint or error

f - Runs until the current function is finished

s - Runs the next line of the program

s N - Runs the next N lines of the program

n - Like s, but it does not step into functions

u N - Runs until you get N lines in front of the current line

p var - Prints the current value of the variable "var"

bt - Prints a stack trace

u - Goes up a level in the stack

d - Goes down a level in the stack

q - Quits gdb