Gem #119: Создание скриптов GDB- Часть 1

Автор: Jean-Charles Delay, AdaCore

Краткое содержание: GDB, или GNU Project Debugger, - весьма удобный инструмент. В общих случаях его применение включает базовые команды CLI - break, run, print, и т.д. Но, кроме этого, он предоставляет доступ к множеству возможностей. Одна из них - создание скриптов. По аналогии с .rc для конфигурации оболочки можно добавить .gdbinit для отладчика. В Gem #119 описываются некоторые доступные возможности скриптов GDB и общая настройка с помощью .gdbinit.

Давайте начнём…

Файл конфигурации GDB находится в домашнем каталоге пользователя:

```
${HOME}/.gdbinit.
```

Когда GDB запускается, он выдает этот файл, если он существует, что означает, что он оценивает все команды в файле, которые могут быть любой из доступных команд CLI. На базовом уровне этот файл можно использовать для простых команд конфигурации, например, для выбора желаемого формата ассемблера по умолчанию или изменения стиля отображения ввода-вывода по умолчанию.

Но GDB предлагает больше: он также может читать язык макро-кодирования, который позволяет более мощные настройки.

Этот язык имеет следующий формат:

```
define _command_
  _code_
end
document _command_
_help text_
End
```

Такие команды называются пользовательскими командами. В пользовательских макросах можно использовать и комбинировать любые стандартные команды интерфейса командной строки GDB.

Раздел документа важен, так как он используется GDB для создания выходных данных команды help при связывании с пользовательскими командами.

Язык GDB также предлагает набор инструкций по управлению потоком и позволяет задавать параметры. Однако манипулирование параметрами ограничено, поскольку GDB предлагает только следующие переменные:

```
$argc
$arg0
$arg1
$arg2
...
```

Однако обратите внимание, что GDB не предоставляет массив, например $argv.

Инструкция `SET`

Вы можете присвоить результат выражения переменной среды с помощью набора инструкции `set`, например:

```
set $VAR = 0
set $byte = *(unsigned char *)$arg0
```

Оператор `if`

```
if _expression_
  _statements_
else
  _statements_
End
```

Оператор `While`

```
while _expression_
  _statements_
End
```

Управляемый вывод

Во время выполнения командного файла или пользовательской команды обычный вывод GDB подавляется; отображается только то, что явно выводится командами в определении.

GDB предоставляет три команды для генерации любого желаемого результата:

```
echo _text_
```

Эта команда печатает `_text_` включая любой непечатаемый символ, которого оставляют в строке C-стиля. Никакая новая строка не печатается, если Вы не определяете тот с помощью `' character`. Можно также использовать `escape`-последовательности для цветов вывода для цветного терминала. Например, `'\033 [01; 31'` то будет печать следующих символов красного цвета, тогда как `'\033 [0m'` сбрасывает цветной вывод, для установки по умолчанию.

```
output _expression_
```

Это печатает значение `_expression_` и ничего, кроме этого значения (нет новой строки, нет `` `$nn = ' ``).

```
printf _string_, _expressions_...
```

Это печатает значения `_expressions_` под управлением строки формата `_string_`. `_expressions_` может быть или числами или указателями.

Например, можно распечатать два значения в шестнадцатеричном числе как это:

```
printf "foo, bar-foo = 0x%x, 0x%x
", foo, bar-foo
```

Вызов `Shell`

Вы можете вызвать оболочку через командный процессор GDB. Если оболочка существует, GDB использует переменную среды $SHELL для определения оболочки для запуска. В противном

случае используется оболочка по умолчанию (`'/bin / sh'` в системах Unix, `` `COMMAND.COM' `` на MS-DOS, etc.).

Вызов `make`

Предположим, что вы скомпилируете свой проект с помощью `Makefile` и `make`-программы. Затем вы можете использовать команду `shell` оболочки GDB для запуска `make`. GDB обычно предоставляет вам собственную команду `make`, которая выполняет программу `make`. Однако для пользователей GNAT Pro Ada программа сборки является `gprbuild`, которая не существует в GDB. Однако вы можете определить команду, которая будет вызывать `gprbuild`, например:

```
define gprbuild
  if $argc == 0
      shell gprbuild
  end
  if $argc == 1
      shell gprbuild $arg0
  end
  if $argc > 1
      help gprbuild
  end
end
document gprbuild
Run the gprbuild program, optionally specifying the project name as
parameter.
Usage: gprbuild [project_name]
End
```

Стиль отображения основания счисления `Radix`

GDB позволяет изменять стиль отображения radix по умолчанию - как для ввода, так и для вывода - позволяя использовать следующие форматы: восьмеричный, десятичный и шестнадцатеричный.

В GDB можно всегда вводить числа в восьмеричном, десятичном или шестнадцатеричном формате по обычным правилам: восьмеричные числа начинаются с '0', десятичные числа заканчиваются ‘.', а шестнадцатеричные числа начинаются с '0x'.

Однако числа, которые не начинаются ни с одного из них, по умолчанию вводятся в базе 10.

Команды radix следующие:

```
set input-radix _base_
set output-radix _base_
```

Внимание: сама база должна быть указана либо однозначно, либо с использованием текущего значения по умолчанию radix.

Таким образом можно выбрать шестнадцатеричный стиль по умолчанию с помощью следующих команд:

```
set input-radix 0x10
set output-radix 0x10
```

Внешний вид приглашения ввода команды

Изменение приглашения также возможно и похоже на изменение командной строки. В следующем примере ваш gdb-запрос на ввод команды ‘gdb $ 'отображается красным:

```
set prompt \033[01;31mgdb$ \033[0m
```

Препятствование тому, чтобы GDB приостановился во время долгого вывода

Возможно, вы уже сталкивались с тем, что, когда GDB нужно напечатать больше строк, чем может отображать высота терминала,он приостанавливается каждый раз, когда консоль заполнена. Чтобы предотвратить это и заставить GDB отображать всю информацию одновременно, используйте следующие настройки:

```
set height 0
set width 0
```

Изменение формата ассемблерного кода

GDB поддерживает различные disassemble форматы наборов инструкций при разборке программы.

Вы можете настроить GDB на использование формата intel или att. Однако эта команда определена только для семейства Intel x86 процессоров:

```
set disassembly-flavor intel
set disassembly-flavor att
```

Набор инструкций по умолчанию - att (AT & T – формат инструкций, используемый по умолчанию ассемблерами Unix для процессоров семейства Intel x86).

Дополнительные сообщения

По умолчанию GDB осторожен и задает много вопросов для подтверждения определенных команд. Если вы готовы непоколебимо столкнуться с последствиями своих собственных команд, вы можете отключить эту " функцию":

```
set confirm off
```

Другие возможности

Для полного списка возможностей GDB см. руководство GDB.

В Gem #120 мы рассмотрим примеры более сложной поддержки сценариев GDB.

Связанный со статьёй текст программы

Attached Files отсутствуют

Файлы примеров Ada Gems распространяются AdaCore и могут быть использованы или изменены для любых целей без ограничений.

Об авторе

Сведения об авторе отсутствуют.

Last Updated: 10/26/2017
Posted on: 2/27/2012

Обсуждение…