Undo™

—

GDB: A Lot More Than You Knew

Greg Law

co-founder and CEO, Undo

The history

I well remember [...] on one of my journeys between the EDSAC room and the punching equipment that "hesitating at the angles of stairs" the realization came over me with full force that a good part of the remainder of my life was going to be spent in finding errors in my own programs.

Sir Maurice Wilkes, 1913-2010

Disclaimer: random bunch of stuff

Learnt along the way, talking to customers

Lots I don't know, lots inevitably missing

    please help me improve these slides!

Most of this is about knowing what you don't know

    `info gdb` is quite a useful manual

GDB - more than you knew

GDB may not be intuitive but it is very powerful

- Easy to use, just not so easy to learn

GDB - more than you knew

GDB may not be intuitive but it is very powerful

- Easy to use, just not so easy to learn

TUI: Text User Interface

- As useful as it is poorly named!

TUI top tips

ctrl-x-a: toggle to/from TUI mode

ctrl-l: refresh the screen

ctrl-p / ctrl-n: prev, next, commands

ctrl-x-2: second window; cycle through

GDB has Python!

GDB has Python!

Full Python interpreter with access to standard modules

      (Unless your gdb installation is messed up!)

The `gdb` python module gives most access to gdb

    `(gdb) python gdb.execute()` to do gdb commands

    `(gdb) python gdb.parse_and_eval()` to get data from inferior

    `(gdb) python help('gdb')` to see online help

Python Pretty Printers

```python
class MyPrinter(object):

    def __init__(self,val):

        self.val = val

    def to_string(self):

        return ( self.val['member'])


import gdb.printing

pp = gdb.printing.RegexpCollectionPrettyPrinter('mystruct')

pp.add_printer('mystruct', '^mystruct$', MyPrinter)

gdb.printing.register_pretty_printer( gdb.current_objfile(), pp)
```

Reversible Debugging - how did that happen?

GDB inbuilt reversible debugging: Works well, but is **very** slow

Reversible Debugging - how did that happen?

GDB inbuilt reversible debugging: Works well, but is **very** slow

GDB in-build 'record btrace': Uses Intel branch trace.

      Not really reversible, no data

      Quite slow

rr: Very good at what it does, though somewhat limited features/platform support

UndoDB: perfect!

Reversible Debugging - how did that happen?

GDB inbuilt reversible debugging: Works well, but is **very** slow

GDB in-build 'record btrace': Uses Intel branch trace.

    Not really reversible, no data

    Quite slow

rr: Very good at what it does, though somewhat limited features/platform support

UndoDB: perfect!

    But expensive :-)

.gdbinit

My ~/.gdbinit is nice and simple:

```
set history save on

set print pretty on

set pagination off

set confirm off
```

If you're funky, it's easy for weird stuff to happen.

Hint: have a project gdbinit with lots of stuff in it, and source that.

Remote debugging

Debug over serial/sockets to a remote server

Start gdbserver localhost:2000 ./a.out

Then connect from a gdb with e.g. 'target remote localhost:2000'

Multiprocess Debugging

Debug multiple 'inferiors' simultaneously

Add new inferiors

Follow fork/exec

Multiprocess Debugging

```
set follow-fork-mode child|parent

set detach-on-fork off

info inferiors

inferior N

set follow-exec-mode new|same

add-inferior <count> <name>

remove-inferior N

clone-inferior

print $_inferior
```

Non-stop mode

Other threads continue while you're at the prompt

Non-stop mode

Other threads continue while you're at the prompt

```
set non-stop on

continue -a
```

Make sure you set pagination off otherwise bad stuff happens!

Breakpoints and watchpoints

`watch foo`   stop when foo is modified

`watch -l foo`   watch location

`rwatch foo`   stop when foo is read

`watch foo thread 3`   stop when thread 3 modifies foo

`watch foo if foo > 10`   stop when foo is > 10

thread apply

```
thread apply 1-4 print $sp

thread apply all backtrace

Thread apply all backtrace full
```

calling inferior functions

`call foo()` will call foo in your inferior

But beware, print may well do too, e.g.

    `print foo()`

    `print foo+bar` (if C++)

    `print errno`

And beware, below will call `strcpy()` *and* `malloc()`!

    `call strcpy( buffer, "Hello, world!\n")`

Dynamic Printf

Use dprintf to put printf's in your code without recompiling, e.g.

```
dprintf mutex_lock,"m is %p m->magic is %u\n",m,m->magic
```

control how the printfs happen:

```
set dprintf-style gdb|call|agent

set dprintf-function fprintf

set dprintf-channel mylog
```

Catchpoints

Catchpoints are like breakpoints but catch certain events, such as C++ exceptions

  e.g. `catch catch` to stop when C++ exceptions are caught

  e.g. `catch syscall nanosleep` to stop at nanosleep system call

  e.g. `catch syscall 100` to stop at system call number 100

More Python

Create your own commands

```python
class my_command( gdb.Command):

    '''doc string'''

    def __init__( self):

        gdb.Command.__init__( self, 'my-command', gdb.COMMAND_NONE)

    def invoke( self, args, from_tty):

        do_bunch_of_python()

my_command()
```

Yet More Python

Hook certain kinds of events

```python
def stop_handler( ev):
    print( 'stop event!')
    if isinstance( ev, gdb.SignalEvent):
        print( 'its a signal: ' + ev.stop_signal)

gdb.events.stop.connect( stop_handler)
```

Other cool things...

- **`tbreak`** — temporary breakpoint
- **`rbreak`** — reg-ex breakpoint
- **`command`** — list of commands to be executed when breakpoint hit
- **`silent`** — special command to suppress output on breakpoint hit
- **`save breakpoints`** — save a list of breakpoints to a script
- **`save history`** — save history of executed gdb commands
- **`info line foo.c:42`** — show PC for line
- **`info line * $pc`** — show line begin/end for current program counter

And finally...

- gcc's -g and -O are orthogonal; gcc -Og is optimised but doesn't mess up debug
- see also gdb dashboard on github

Thank you

Undo

9 Signet Court, Swann's Road,
Cambridge, CB5 8LA, UK

For a free 30 day trial of UndoDB,
visit our website, or contact
sales@undo.io for more information
about our products.

Undo™