

GDB基础

张银奎

2017/2/14珠海白石富华里

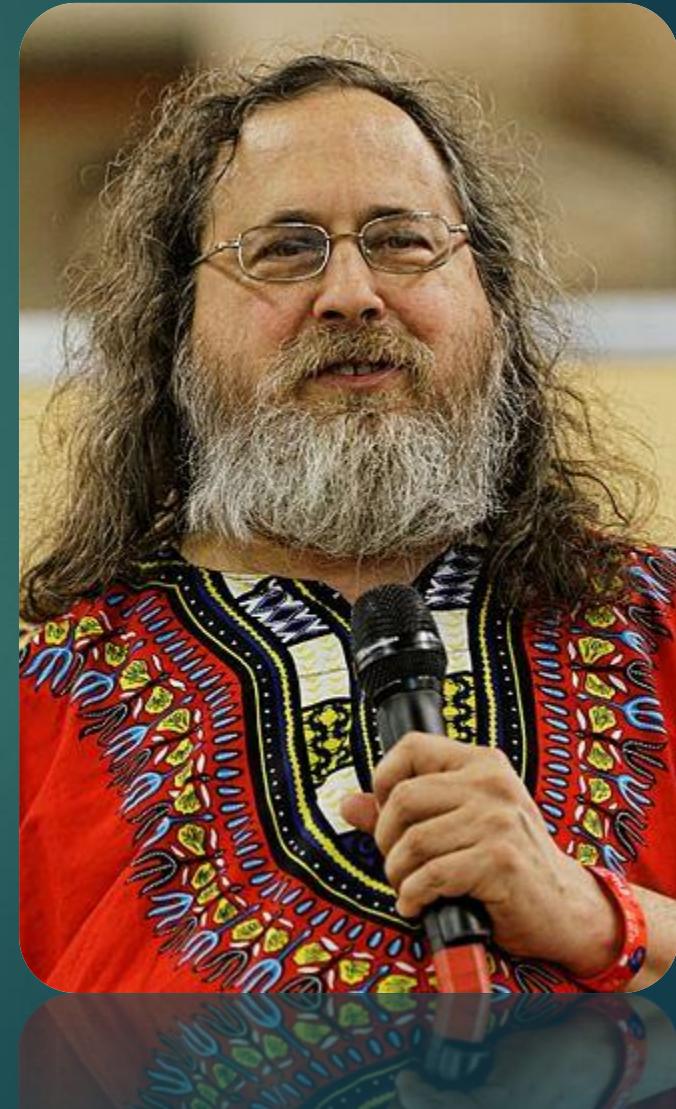


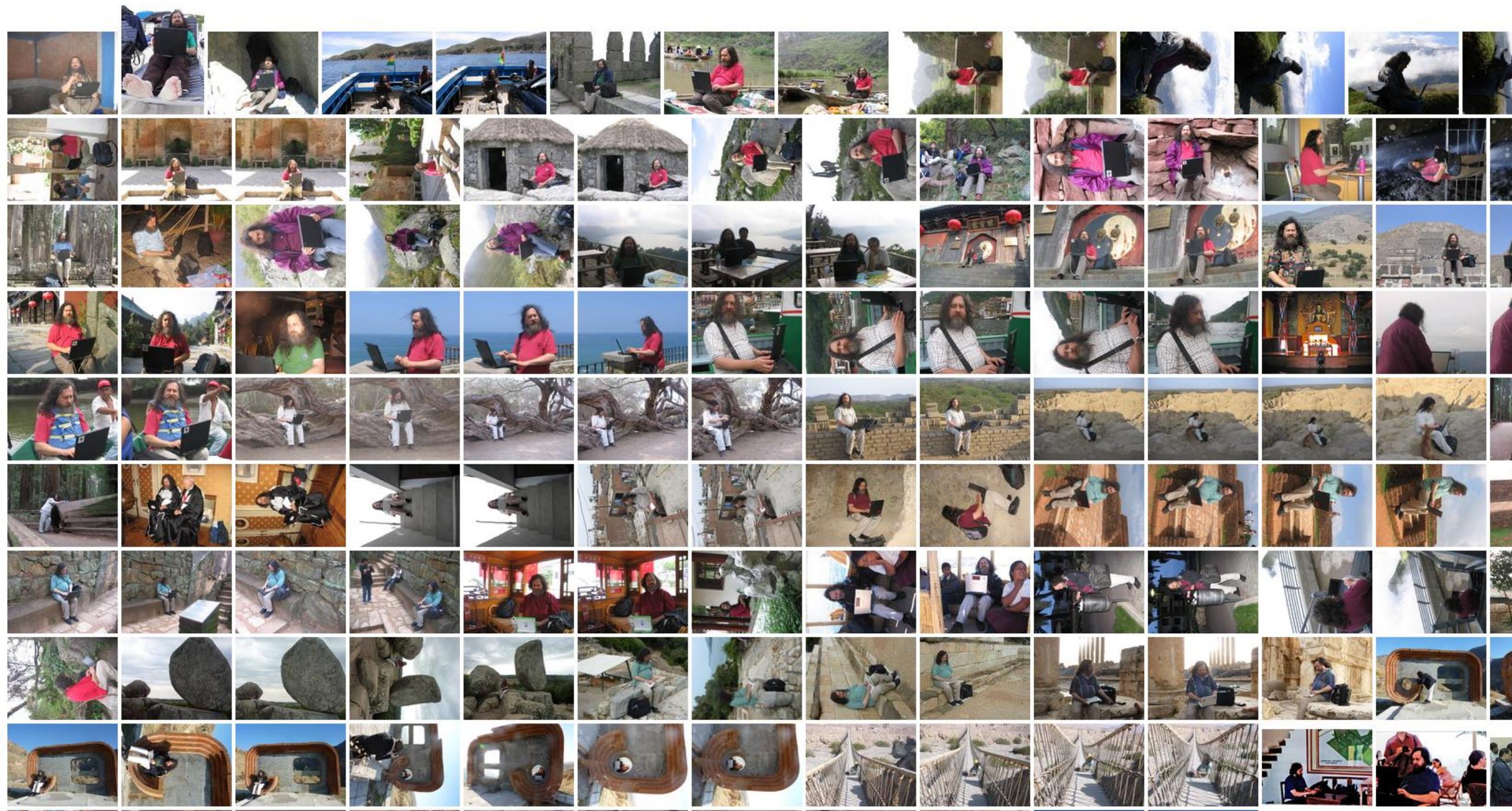
Richard Matthew Stallman

“Richard Matthew Stallman is a software developer and software freedom activist. Born in 1953, he attended Harvard starting in 1970 and graduated in 1974 with a Bachelor of Arts in physics. From September 1974 to June 1975 he was a graduate student in physics at MIT.”

Stallman developed a number of widely used software components of the GNU system: the GNU Compiler Collection, the GNU symbolic debugger (gdb), GNU Emacs, and various others.

- ▶ <https://www.stallman.org/>





RMS in China, 2000



Beijing -> Chengdu -> JiuzhaiGou -> Wuhan and to Shanghai"



Richard Stallman using his Lemote machine at Indian Institute of Technology Madras, Chennai before his lecture on 'Free Software, Freedom and Education' organized by Free Software Foundation, Tamil Nadu.

https://en.wikipedia.org/wiki/Richard_Stallman

What hardware do you use?

I am using a Lemote Yeelong, a netbook with a Loongson chip and a 9-inch display. This is my only computer, and I use it all the time. I chose it because I can run it with 100% free software even at the BIOS level.

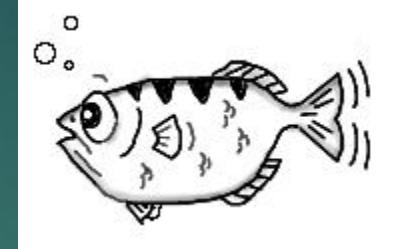
<https://usesthis.com/interviews/richard.stallman/>

GNU Projects

- ▶ Stallman announced the plan for the GNU operating system in September 1983 on several ARPANET mailing lists and USENET.
- ▶ Stallman was responsible for contributing many necessary tools, including a text editor (Emacs), compiler (GCC), debugger (GNU Debugger), and a build automator (GNU make).
- ▶ In 1991, Linus Torvalds, a Finnish student, used the GNU's development tools to produce the free monolithic Linux kernel.

- ▶ Man ls
 - ▶ AUTHOR
- ▶ Written by Richard M. Stallman and David MacKenzie.

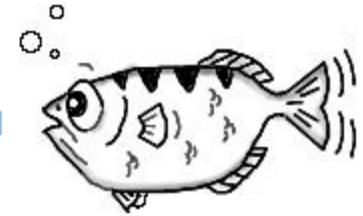
GDB



- ▶ GNU Debugger
 - ▶ 1986年, Richard Stallman创建
 - ▶ 1990-1993, John Gilmore维护
 - ▶ 目前在GDB Steering Committee
 - ▶ It was modeled after the DBX debugger, which came with Berkeley Unix distributions
 - 支持很多种CPU architecture
 - ▶ A29K, ARC, ETRAX CRIS, D10V, D30V, FR-30, FR-V, Intel i960, M32R, 68HC11, Motorola 88000, MCORE, MN10200, MN10300, NS32K, Stormy16, V850, Z8000 **and many more**
 - 默认为命令行界面,有很多GUI的前端(Add-on)

GDB: The GNU Project Debugger

[\[bugs\]](#) [\[GDB Maintainers\]](#) [\[contributing\]](#) [\[current git\]](#) [\[documentation\]](#) [\[download\]](#) [\[home\]](#) [\[irc\]](#) [\[links\]](#) [\[mailing lists\]](#) [\[news\]](#) [\[schedule\]](#) [\[song\]](#)
[\[wiki\]](#)



GDB: The GNU Project Debugger

What is GDB?

GDB, the GNU Project debugger, allows you to see what is going on 'inside' another program while it executes -- or what another program was doing at the moment it crashed.

GDB can do four main kinds of things (plus other things in support of these) to help you catch bugs in the act:

- Start your program, specifying anything that might affect its behavior.
- Make your program stop on specified conditions.
- Examine what has happened, when your program has stopped.
- Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another.

The program being debugged can be written in Ada, C, C++, Objective-C, Pascal (and many other languages). Those programs might be executing on the same machine as GDB (native) or on another machine (remote). GDB can run on most popular UNIX and Microsoft Windows variants.

GDB version 7.11.1

Version [7.11.1](#) of GDB, the GNU Debugger, is now available for [download](#). See the [ANNOUNCEMENT](#) for details including changes in this release.

► <https://www.gnu.org/software/gdb/>

Contributors to gdb

Contributors to GDB

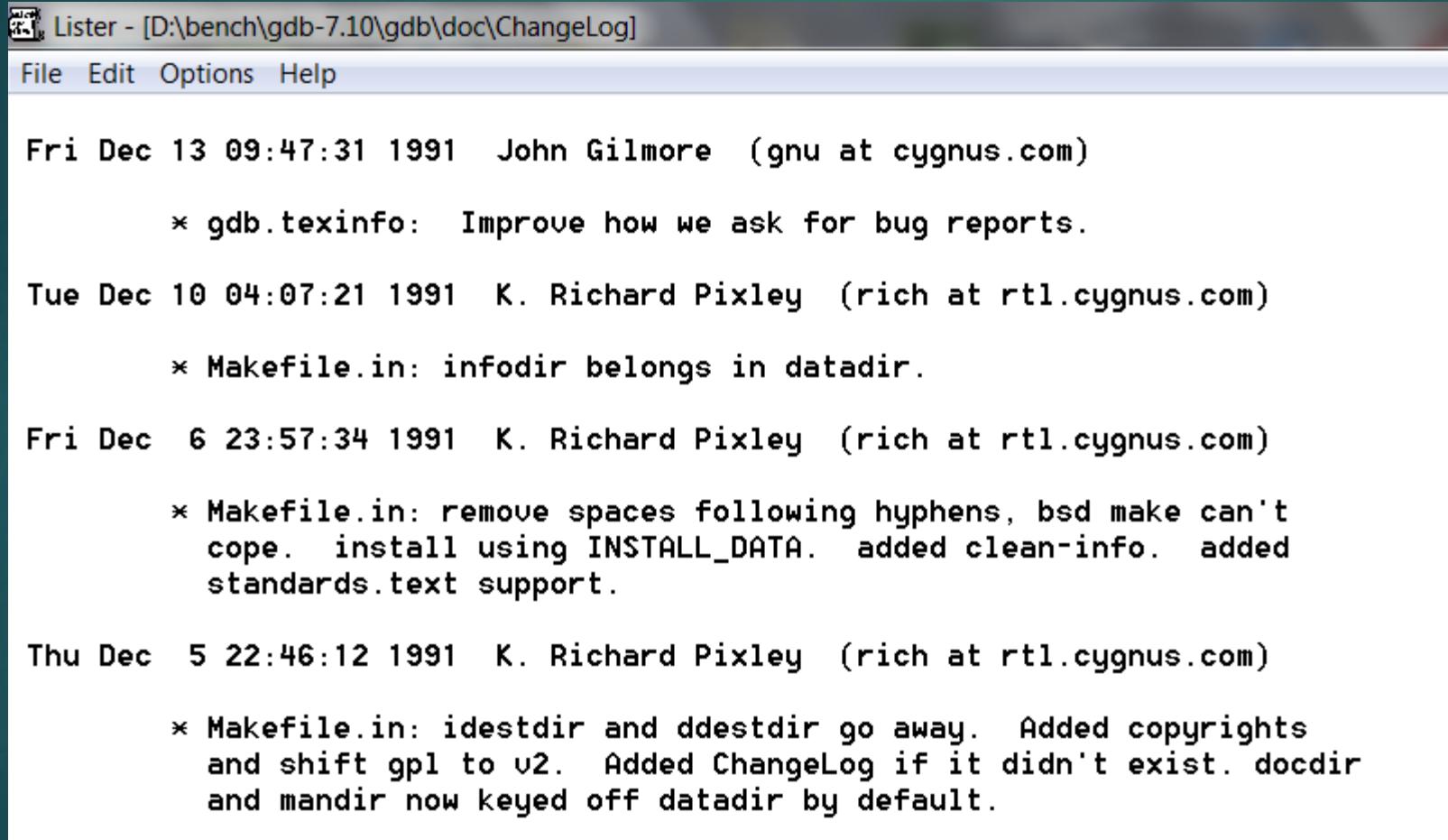
Richard Stallman was the original author of GDB, and of many other GNU programs. Many others have contributed to its development. This section attempts to credit major contributors. One of the virtues of free software is that everyone is free to contribute to it; with regret, we cannot actually acknowledge everyone here. The file ‘[ChangeLog](#)’ in the GDB distribution approximates a blow-by-blow account.

Changes much prior to version 2.0 are lost in the mists of time.

Plea: Additions to this section are particularly welcome. If you or your friends (or enemies, to be evenhanded) have been unfairly omitted from this list, we would like to add your names!

So that they may not regard their many labors as thankless, we particularly thank those who shepherded GDB through major releases: Andrew Cagney (releases 6.3, 6.2, 6.1, 6.0, 5.3, 5.2, 5.1 and 5.0); Jim Blandy (release 4.18); Jason Molenda (release 4.17); Stan Shebs (release 4.14); Fred Fish (releases 4.16, 4.15, 4.13, 4.12, 4.11, 4.10, and 4.9); Stu Grossman and John Gilmore (releases 4.8, 4.7, 4.6, 4.5, and 4.4); John Gilmore (releases 4.3, 4.2, 4.1, 4.0, and 3.9); Jim Kingdon (releases 3.5, 3.4, and 3.3); and Randy Smith (releases 3.2, 3.1, and 3.0).

ChangeLog



```
File Edit Options Help

Fri Dec 13 09:47:31 1991 John Gilmore (gnu at cygnus.com)
  * gdb.texinfo: Improve how we ask for bug reports.

Tue Dec 10 04:07:21 1991 K. Richard Pixley (rich at rtl.cygnus.com)
  * Makefile.in: infodir belongs in datadir.

Fri Dec  6 23:57:34 1991 K. Richard Pixley (rich at rtl.cygnus.com)
  * Makefile.in: remove spaces following hyphens, bsd make can't
    cope. install using INSTALL_DATA. added clean-info. added
    standards.text support.

Thu Dec  5 22:46:12 1991 K. Richard Pixley (rich at rtl.cygnus.com)
  * Makefile.in: idestdir and ddestdir go away. Added copyrights
    and shift gpl to v2. Added ChangeLog if it didn't exist. docdir
    and mandir now keyed off datadir by default.
```

/gdb/README

README for GDB release

This is GDB, the GNU source-level debugger.

A summary of new features is in the file `gdb/NEWS'.

Check the GDB home page at <http://www.gnu.org/software/gdb/> for up to date release information, mailing list links and archives, etc.

The file `gdb/PROBLEMS' contains information on problems identified late in the release cycle. GDB's bug tracking data base at <http://www.gnu.org/software/gdb/bugs/> contains a more complete list of bugs.

RMS's gdb Debugger Tutorial

RMS's gdb Debugger Tutorial

"Don't worry if it doesn't work right. If everything did,
you'd be out of a job."

Unknown

gdb Debugger Frequently Asked Questions

Table of Contents

1. How do I use the gdb debugger?

How do I...

1. [compile with debugging symbols?](#)
2. [run programs with the debugger?](#)
3. [restart a program running in the debugger?](#)
4. [exit the debugger?](#)
5. [get help on debugger commands?](#)

2. How do I watch the execution of my program?

How do I...

1. [stop execution](#)
2. [continue execution](#)

► <http://www.unknownroad.com/rtfm/gdbtut/gdbtoc.html>

Document

Debugging with GDB

The GNU Source-Level Debugger

Tenth Edition, for GDB version 7.11.1

(GDB)

Richard Stallman, Roland Pesch, Stan Shebs, et al.

- ▶ 学习GDB的最好教材
- ▶ Tenth Edition, for gdb
version 7.11.1
- ▶ 33章，多个附录，790页

概览

- 1 A Sample gdb Session
- 2 Getting In and Out of gdb
 - 2.1 Invoking gdb
 - 2.1.1 Choosing Files
 - 2.1.2 Choosing Modes
 - 2.1.3 What gdb Does During Startup
 - 2.2 Quitting gdb
 - 2.3 Shell Commands
 - 2.4 Logging Output
- 3 gdb Commands
 - 3.1 Command Syntax
 - 3.2 Command Completion
 - 3.3 Getting Help

在GDB下运行程序

4 Running Programs Under gdb

4.1 Compiling for Debugging

4.2 Starting your Program

4.3 Your Program's Arguments

4.4 Your Program's Environment

4.5 Your Program's Working Directory

4.6 Your Program's Input and Output

4.7 Debugging an Already-running Process

4.8 Killing the Child Process

4.9 Debugging Multiple Inferiors and Programs

4.10 Debugging Programs with Multiple Threads

4.11 Debugging Forks

4.12 Setting a Bookmark to Return to Later

4.12.1 A Non-obvious Benefit of Using Checkpoints

停止和继续

5 Stopping and Continuing

5.1 Breakpoints, Watchpoints, and Catchpoints

5.1.1 Setting Breakpoints

5.1.2 Setting Watchpoints

5.1.3 Setting Catchpoints

5.1.4 Deleting Breakpoints

5.1.5 Disabling Breakpoints

5.1.6 Break Conditions

5.1.7 Breakpoint Command Lists

5.1.8 Dynamic Printf

5.1.9 How to save breakpoints to a file

5.1.10 Static Probe Points

5.1.11 “Cannot insert breakpoints”

5.1.12 “Breakpoint address adjusted...”

5.2 Continuing and Stepping

5.3 Skipping Over Functions and Files

5.4 Signals

5.5 Stopping and Starting Multi-thread Programs

5.5.1 All-Stop Mode

5.5.2 Non-Stop Mode

5.5.3 Background Execution

5.5.4 Thread-Specific Breakpoints

5.5.5 Interrupted System Calls

5.5.6 Observer Mode

炫酷功能

6 Running programs backward

7 Recording Inferior's Execution and Replaying It

查看栈、源代码

8 Examining the Stack

 8.1 Stack Frames

 8.2 Backtraces

 8.3 Selecting a Frame

 8.4 Information About a Frame

 8.5 Management of Frame Filters.

9 Examining Source Files

 9.1 Printing Source Lines

 9.2 Specifying a Location

 9.2.1 Linespec Locations

 9.2.2 Explicit Locations

 9.2.3 Address Locations

 9.3 Editing Source Files

 9.3.1 Choosing your Editor

 9.4 Searching Source Files

 9.5 Specifying Source Directories

 9.6 Source and Machine Code

查看数据

10 Examining Data

10.1 Expressions

10.2 Ambiguous Expressions

10.3 Program Variables

10.4 Artificial Arrays

10.5 Output Formats

10.6 Examining Memory

10.7 Automatic Display

10.8 Print Settings

10.9 Pretty Printing

10.9.1 Pretty-Printer Introduction

10.9.2 Pretty-Printer Example

10.9.3 Pretty-Printer Commands

10.10 Value History

10.11 Convenience Variables

10.12 Convenience Functions

10.13 Registers

10.14 Floating Point Hardware

10.15 Vector Unit

10.16 Operating System Auxiliary Information

10.17 Memory Region Attributes

10.17.1 Attributes

10.17.1.1 Memory Access Mode

10.17.1.2 Memory Access Size

10.17.1.3 Data Cache

10.17.2 Memory Access Checking

10.18 Copy Between Memory and a File

10.19 How to Produce a Core File from Your Program

10.20 Character Sets

10.21 Caching Data of Targets

10.22 Search Memory

10.23 Value Sizes

跟踪点

11 Debugging Optimized Code

11.1 Inline Functions

11.2 Tail Call Frames

12 C Preprocessor Macros

13 Tracepoints

13.1 Commands to Set Tracepoints

13.1.1 Create and Delete Tracepoints

13.1.2 Enable and Disable Tracepoints

13.1.3 Tracepoint Passcounts

13.1.4 Tracepoint Conditions

13.1.5 Trace State Variables

13.1.6 Tracepoint Action Lists

13.1.7 Listing Tracepoints

13.1.8 Listing Static Tracepoint Markers

13.1.9 Starting and Stopping Trace Experiments

13.1.10 Tracepoint Restrictions

13.2 Using the Collected Data

13.2.1 tfind n

13.2.2 tdump

13.2.3 save tracepoints filename

13.3 Convenience Variables for Tracepoints

13.4 Using Trace Files

查看符号表和改变执行目标

16 Examining the Symbol Table

17 Altering Execution

17.1 Assignment to Variables

17.2 Continuing at a Different Address

17.3 Giving your Program a Signal

17.4 Returning from a Function

17.5 Calling Program Functions

17.6 Patching Programs

17.7 Compiling and injecting code in gdb

17.7.1 Compilation options for the compile command

17.7.2 Caveats when using the compile command

17.7.3 Compiler search for the compile command

指定文件和控制GDB

18 gdb Files

- 18.1 Commands to Specify Files
- 18.2 File Caching
- 18.3 Debugging Information in Separate Files
- 18.4 Debugging information in a special section
- 18.5 Index Files Speed Up gdb
- 18.6 Errors Reading Symbol Files
- 18.7 GDB Data Files

22 Controlling gdb

- 22.1 Prompt
- 22.2 Command Editing
- 22.3 Command History
- 22.4 Screen Size
- 22.5 Numbers
- 22.6 Configuring the Current ABI
- 22.7 Automatically loading associated files
 - 22.7.1 Automatically loading init file in the current directory
 - 22.7.2 Automatically loading thread debugging library
 - 22.7.3 Security restriction for auto-loading
 - 22.7.4 Displaying files tried for auto-load
- 22.8 Optional Warnings and Messages
- 22.9 Optional Messages about Internal Happenings
- 22.10 Other Miscellaneous Settings



Syntax

- ▶ A gdb command is a single line of input. There is no limit on how long it can be.
- ▶ You can **repeat** certain gdb commands by typing just **RET**.
- ▶ You can also use the TAB key to get gdb to fill out the rest of a word in a command
- ▶ # comment

```
(gdb) b make_ TAB
GDB sounds bell; press TAB again, to see:
make_a_section_from_file      make_environ
make_abs_section                make_function_type
make_blockvector                 make_pointer_type
make_cleanup                     make_reference_type
make_command                     make_symbol_completion_list
(gdb) b make_
```

缩写和别名

- ▶ You can abbreviate a gdb command to the first few letters of the command name, if that abbreviation is unambiguous;
 - ▶ (GDB) info > i
 - ▶ (GDB) continue > c
- ▶ 很多命令有简单的别名
 - ▶ (GDB) backtrace > bt
 - ▶ (GDB) ptype > pt

Convenience Variables

- ▶ GDB维护的违变量
- ▶ 以\$开头
- ▶ \$ most recent displayed value
- ▶ \$n nth displayed value
- ▶ \$\$ displayed value previous to \$
- ▶ \$\$n nth displayed value back from \$
- ▶ 可以使用set命令定义

```
set $foo = *object_ptr
```

```
(gdb) p argv[0]
$4 = 0xbffff567 "/home/ge/work/llaolao3/baner"
(gdb) p $
$5 = 0xbffff567 "/home/ge/work/llaolao3/baner"
```

```
(gdb) show conv
$_thread = 2
$_siginfo = {si_signo = 19, si_errno = 0, si_code = 0, _sifields = {_pad = {0,
    0, 135725706, 162241784, -1220761920, 0, 0, 162241784, 138194409,
    -1248857920, -1220759564, -1220756960, 1, 1, -1222068232, -1220756960,
    -1080018849, 1, 134858900, 160600224, 152099856, -1248857920,
    -1224747328, -1220759564, -1, 152244240, -1080018072, -1080017996,
    -1080018184}, _kill = {si_pid = 0, si_uid = 0}, _timer = {si_tid = 0,
    si_overrun = 0, si_sigval = {sival_int = 135725706,
        sival_ptr = 0x817028a}}, _rt = {si_pid = 0, si_uid = 0, si_sigval = {
            sival_int = 135725706, sival_ptr = 0x817028a}}, _sigchld = {si_pid = 0,
    si_uid = 0, si_status = 135725706, si_utime = 162241784,
    si_stime = -1220761920}, _sigfault = {si_addr = 0x0}, _sigpoll = {
        si_band = 0, si_fd = 0}}}
$_sdata = void
```

`$_`和`$__`

- ▶ The variable `$_` is automatically set by the `x` command to the last address examined.
- ▶ The variable `$__` is automatically set by the `x` command to the value found in the last address examined. Its type is chosen to match the format in which the data was printed

```
(gdb) x argv
0xbfffff7f4:    0x24
(gdb) p $_
$22 = 36
(gdb) p $_
$23 = (int8_t *) 0xbfffff7f4
```

`$_thread`

- ▶ Gdb provides two convenience variables, `$_thread` and `$_gthread` (the latter being pretty new), which can be used in conditions to refer to the current thread.
- ▶ So, once the worker thread has started, you can use `info thread` to find its number. Then you can change your breakpoint (supposing for this example that it is breakpoint 2) like:
- ▶ (gdb) cond 2 `$_thread != 57`

Gdb variables

- ▶ (gdb) set \$foo = 4
- ▶ (gdb) p \$foo
- ▶ \$3 = 4

Register Variables

- ▶ (gdb) break write if \$rsi == 2

表达式归纳

- ▶ expr an expression in C, C++, or Modula-2(including function calls), or:
- ▶ addr@len an array of len elements beginning at addr
- ▶ file::nm a variable or function nm defined in file
- ▶ **{type}addr** read memory at addr as specied type
- ▶ \$ most recent displayed value
- ▶ \$n nth displayed value
- ▶ \$\$ displayed value previous to \$
- ▶ \$\$n nth displayed value back from \$
- ▶ \$_ last address examined with x
- ▶ \$__ value at address \$_
- ▶ \$var convenience variable; assign any value
- ▶ show values [n] show last 10 values [or surrounding \$n]
- ▶ show conv display all convenience variables

```
(gdb) p argv[0]@2
$7 = {0xbffff567 "/home/ge/work/llaolao3/baner", 0xbffff584 "0"}
(gdb) p $$
$8 = 0xbffff567 "/home/ge/work/llaolao3/baner"
```

```
(gdb) p {int}argv
$14 = -1073744537
(gdb) p /x {int}argv
$15 = 0xbffff567
```

Shell Commands

- ▶ shell command-string
- ▶ !command-string

```
(gdb) !dir
Volume in drive C is OSDisk
Volume Serial Number is 1E62-585F

Directory of C:\TEMP\ge

10/08/2016  07:25 AM    <DIR>      .
10/08/2016  07:25 AM    <DIR>      ..
09/01/2016  10:12 PM    <DIR>      cmos
08/15/2016  08:30 PM            28,175 dev.txt
09/21/2016  11:16 AM            11,587,305 fedcore.ko
09/30/2016  12:01 PM            3,936,768 fedtest.frames.txt
09/30/2016  09:55 AM            1,169,286 fedtest.ko
08/12/2016  05:42 PM            18,934,392 gdb-7.11.tar.xz
08/12/2016  05:40 PM            4,993,536 gdb.exe
01/13/2015  10:00 AM            2,650,119 gdb.zip
08/22/2016  09:59 PM            5,326,498,816 ge32.ova
08/17/2016  10:01 PM            107,779 laolao.elf.txt
09/28/2016  01:04 PM    <DIR>      lib
08/14/2016  12:15 PM    <DIR>      linux-3.12.2
09/20/2016  11:21 AM            2,680,575,786 linux_dwarf.txt
09/01/2016  10:12 PM    <DIR>      llaolao
09/07/2016  08:09 PM    <DIR>      llaolao2
09/18/2016  10:50 PM    <DIR>      llaolao3
10/08/2016  07:25 AM            0 readme.txt
08/18/2016  04:57 PM    <DIR>      syseye
09/28/2016  10:53 AM            2,778 vdso.frames
08/14/2016  12:15 PM            217,010,778 vmlinuz-3.12.2.ko
13 File(s)   8,267,495,518 bytes
9 Dir(s)    3,873,128,448 bytes free
```

Info and show

- ▶ info -- Generic command for showing things about the program being debugged
- ▶ show -- Generic command for showing things about the debugger

```
(gdb) show version
GNU gdb (GDB) 7.9
Copyright (C) 2015 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-pc-mingw32".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
This binary was built by Equation Solution <http://www.Equation.com>.
```

Debug the debugger

```
(gdb) show debug
arch: Architecture debugging is 0.
check-physname: Whether to check "physname" is off.
displaced: Displace stepping debugging is off.
dwarf2-die: Debugging of the dwarf2 DIE reader is 0.
entry-values: Entry values and tail call frames debugging is 0.
expression: Expression debugging is 0.
frame: Frame debugging is 0.
infrun: Inferior debugging is 0.
jit: JIT debugging is 0.
libthread-db: libthread-db debugging is 0.
lin-lwp: Debugging of GNU/Linux lwp module is 0.
observer: Observer debugging is 0.
overload: Debugging of C++ overloading is 0.
parser: Parser debugging is off.
record: Debugging of process record target is 0.
remote: Debugging of remote protocol is 0.
serial: Serial debugging is 0.
target: Target debugging is 0.
timestamp: Timestamping debugging messages is off.
xml: XML debugging is off.
```

- ▶ set debug -- Generic command for setting gdb debugging flags

Logging Output

`set logging on`

Enable logging.

`set logging off`

Disable logging.

`set logging file file`

Change the name of the current logfile. The default logfile is ‘`gdb.txt`’.

`set logging overwrite [on|off]`

By default, GDB will append to the logfile. Set `overwrite` if you want `set logging on` to overwrite the logfile instead.

`set logging redirect [on|off]`

By default, GDB output will go to both the terminal and the logfile. Set `redirect` if you want output to go only to the log file.

`show logging`

Show the current values of the logging settings.

常用调试命令对照表

WinDBG命令	GDB命令	功能
bp	break或b	设置软件断点
ba	watch	设置硬件断点
k	backtrace或bt	显示函数调用序列（栈回溯）
g	continue或c	恢复执行
p/t	next/step或n/s	单步跟踪
d	x	观察内存
dv	info locals	观察局部变量
dt	pt	观察数据类型（结构）
gu	finish	执行到函数返回
.frame	frame	切换当前栈帧
lm	i shared	列模块

更多常用命令

- ▶ 命令行
 - ▶ run xxx
 - ▶ set args xx xx
 - ▶ show args
- ▶ 观察类型
 - ▶ ptype
 - ▶ whatis
 - ▶ print v@10
- ▶ 管理断点
 - ▶ info/disable/delete break
- ▶ 源代码
 - ▶ list 3,8
 - ▶ info line/source/sources
- ▶ show conv

Essential Commands

gdb program [core] debug program [using coredump core]
b [file:]function set breakpoint at function [in file]
run [arglist] start your program [with arglist]
bt backtrace: display program stack
p expr display the value of an expression
c continue running your program
n next line, stepping over function calls
s next line, stepping into function calls

Starting GDB

gdb start GDB, with no debugging files
gdb program begin debugging *program*
gdb program core debug core dump *core* produced by *program*
gdb --help describe command line options

Stopping GDB

quit exit GDB; also **q** or EOF (eg C-d)
INTERRUPT (eg C-c) terminate current command, or send to running process

Getting Help

help list classes of commands
help class one-line descriptions for commands in *class*
help command describe *command*

Executing your Program

run arglist start your program with *arglist*
run start your program with current argument list
run ... <inf>>outf start your program with input, output redirected
kill kill running program

tty dev use *dev* as stdio and stdout for next run
set args arglist specify *arglist* for next run
set args specify empty argument list
show args display argument list

show env show all environment variables
show env var show value of environment variable *var*
set env var string set environment variable *var*
unset env var remove *var* from environment

Shell Commands

cd dir change working directory to *dir*
pwd Print working directory
make ... call "make"
shell cmd execute arbitrary shell command string

Breakpoints and Watchpoints

break [file:]line set breakpoint at *line* number [in *file*]
b [file:]line eg: **break main.c:37**
break [file:]func set breakpoint at *func* [in *file*]
break +offset set break at *offset* lines from current stop
break -offset
break *addr set breakpoint at address *addr*
break set breakpoint at next instruction
break ... if expr break conditionally on nonzero *expr*
cond n [expr] new conditional expression on breakpoint *n*; make unconditional if no *expr*
tbreak ...
rbreak regex
watch expr
catch event break on all functions matching *regex*
info break show defined breakpoints
info watch show defined watchpoints
clear delete breakpoints at next instruction
clear [file:]fun delete breakpoints at entry to *fun()*
clear [file:]line delete breakpoints on source line
delete [n] delete breakpoints [or breakpoint *n*]
disable [n] disable breakpoints [or breakpoint *n*]
enable [n] enable breakpoints [or breakpoint *n*]
enable once [n] enable breakpoints [or breakpoint *n*]; disable again when reached
enable del [n] enable breakpoints [or breakpoint *n*]; delete when reached
ignore n count ignore breakpoint *n*, *count* times
commands n [silent] command-list execute GDB *command-list* every time breakpoint *n* is reached. [*silent* suppresses default display]
end end of *command-list*

Program Stack

backtrace [n] print trace of all frames in stack; or of *n* frames—innermost if *n*>0, outermost if *n*<0
bt [n]
frame [n] select frame number *n* or frame at address *n*; if no *n*, display current frame
up n select frame *n* frames up
down n select frame *n* frames down
info frame [addr] describe selected frame, or frame at *addr*
info args arguments of selected frame
info locals local variables of selected frame
info reg [rn]... register values [for regs *rn*] in selected frame; *all-reg* includes floating point
info all-reg [rn]

Execution Control

continue [count] continue running; if *count* specified, ignore this breakpoint next *count* times
c [count]
step [count] execute until another line reached; repeat *count* times if specified
s [count]
stepli [count] step by machine instructions rather than source lines
si [count]
next [count] execute next line, including any function calls
n [count]
nexti [count] next machine instruction rather than source line
ni [count]
until [location] run until next instruction (or *location*)
finish run until selected stack frame returns
return [expr] pop selected stack frame without executing [setting return value]
signal num resume execution with signal *s* (none if 0)
jump line resume execution at specified *line* number
jump *address evaluate *expr* without displaying it; use for altering program variables

Display

print [/f] [expr] show value of *expr* [or last value \$] according to format *f*
p [/f] [expr]
x hexadecimal
d signed decimal
u unsigned decimal
o octal
t binary
a address, absolute and relative
c character
f floating point
call [/f] expr like **print** but does not display void
x [/Nuf] expr examine memory at address *expr*; optional format spec follows slash
N count of how many units to display
u unit size; one of
 b individual bytes
 h halfwords (two bytes)
 w words (four bytes)
 g giant words (eight bytes)
f printing format. Any **print** format, or
 s null-terminated string
 i machine instructions
disassem [addr] display memory as machine instructions

Automatic Display

display [/f] expr show value of *expr* each time program stops [according to format *f*]
display
undisplay n display all enabled expressions on list
remove number(s) n from list of automatically displayed expressions
disable disp n
enable disp n
info display numbered list of display expressions

Expressions

<code>expr</code>	an expression in C, C++, or Modula-2 (including function calls), or:
<code>addr@len</code>	an array of <code>len</code> elements beginning at <code>addr</code>
<code>file::nm</code>	a variable or function <code>nm</code> defined in <code>file</code>
<code>{type}addr</code>	read memory at <code>addr</code> as specified <code>type</code>
<code>\$</code>	most recent displayed value
<code>\$n</code>	<code>n</code> th displayed value
<code>\$\$n</code>	displayed value previous to <code>\$n</code>
<code>\$_</code>	<code>n</code> th displayed value back from <code>\$</code>
<code>\$_-</code>	last address examined with <code>x</code>
<code>\$var</code>	value at address <code>\$_-</code>
<code>show values [n]</code>	convenience variable; assign any value
<code>show conv</code>	show last 10 values [or surrounding <code>\$n</code>] display all convenience variables

Symbol Table

<code>info address s</code>	show where symbol <code>s</code> is stored
<code>info func [regex]</code>	show names, types of defined functions (all, or matching <code>regex</code>)
<code>info var [regex]</code>	show names, types of global variables (all, or matching <code>regex</code>)
<code>whatis [expr]</code>	show data type of <code>expr</code> [or <code>\$</code>] without evaluating; <code>ptype</code> gives more detail
<code>ptype [expr]</code>	<code>ptype</code> type, struct, union, or enum

GDB Scripts

<code>source script</code>	read, execute GDB commands from file <code>script</code>
<code>define cmd command-list</code>	create new GDB command <code>cmd</code> ; execute script defined by <code>command-list</code>
<code>end</code>	end of <code>command-list</code>
<code>document cmd help-text</code>	create online documentation for new GDB command <code>cmd</code>
<code>end</code>	end of <code>help-text</code>

Signals

<code>handle signal act</code>	specify GDB actions for <code>signal</code> :
<code>print</code>	announce signal
<code>noprint</code>	be silent for signal
<code>stop</code>	halt execution on signal
<code>nostop</code>	do not halt execution
<code>pass</code>	allow your program to handle signal
<code>nopass</code>	do not allow your program to see signal
<code>info signals</code>	show table of signals, GDB action for each

Debugging Targets

<code>target type param</code>	connect to target machine, process, or file
<code>help target</code>	display available targets
<code>attach param</code>	connect to another process
<code>detach</code>	release target from GDB control

Controlling GDB

<code>set param value</code>	set one of GDB's internal parameters
<code>show param</code>	display current setting of parameter
Parameters understood by <code>set</code> and <code>show</code> :	
<code>complaint limit</code>	number of messages on unusual symbols
<code>confirm on/off</code>	enable or disable cautionary queries
<code>editing on/off</code>	control readline command-line editing
<code>height lpp</code>	number of lines before pause in display
<code>language lang</code>	Language for GDB expressions (auto, c or modula-2)
<code>listsize n</code>	number of lines shown by <code>list</code>
<code>prompt str</code>	use <code>str</code> as GDB prompt
<code>radix base</code>	octal, decimal, or hex number representation
<code>verbose on/off</code>	control messages when loading symbols
<code>width cpl</code>	number of characters before line folded
<code>write on/off</code>	Allow or forbid patching binary, core files (when reopened with <code>exec</code> or <code>core</code>)
<code>history ...</code>	groups with the following options:
<code>h ...</code>	
<code>h exp off/on</code>	disable/enable readline history expansion
<code>h file filename</code>	file for recording GDB command history
<code>h size size</code>	number of commands kept in history list
<code>h save off/on</code>	control use of external file for command history
<code>print ...</code>	groups with the following options:
<code>p ...</code>	
<code>p address on/off</code>	print memory addresses in stacks, values
<code>p array off/on</code>	compact or attractive format for arrays
<code>p demangl on/off</code>	source (demangled) or internal form for C++ symbols
<code>p asm-dem on/off</code>	demangle C++ symbols in machine-instruction output
<code>p elements limit</code>	number of array elements to display
<code>p object on/off</code>	print C++ derived types for objects
<code>p pretty off/on</code>	struct display: compact or indented
<code>p union on/off</code>	display of union members
<code>p vtbl off/on</code>	display of C++ virtual function tables
<code>show commands</code>	show last 10 commands
<code>show commands n</code>	show 10 commands around number <code>n</code>
<code>show commands +</code>	show next 10 commands
Working Files	
<code>file [file]</code>	use <code>file</code> for both symbols and executable; with no arg, discard both
<code>core [file]</code>	read <code>file</code> as coredump; or discard
<code>exec [file]</code>	use <code>file</code> as executable only; or discard
<code>symbol [file]</code>	use symbol table from <code>file</code> ; or discard
<code>load file</code>	dynamically link <code>file</code> and add its symbols
<code>add-sym file addr</code>	read additional symbols from <code>file</code> , dynamically loaded at <code>addr</code>
<code>info files</code>	display working files and targets in use
<code>path dirs</code>	add <code>dirs</code> to front of path searched for executable and symbol files
<code>show path</code>	display executable and symbol file path
<code>info share</code>	list names of shared libraries currently loaded

Source Files

<code>dir names</code>	add directory <code>names</code> to front of source path
<code>dir</code>	clear source path
<code>show dir</code>	show current source path
<code>list</code>	show next ten lines of source
<code>list -</code>	show previous ten lines
<code>list lines</code>	display source surrounding <code>lines</code> , specified as:
<code>[file:]num</code>	line number [in named file]
<code>[file:]function</code>	beginning of function [in named file]
<code>+off</code>	<code>off</code> lines after last printed
<code>-off</code>	<code>off</code> lines previous to last printed
<code>*address</code>	line containing <code>address</code>
<code>list f,l</code>	from line <code>f</code> to line <code>l</code>
<code>info line num</code>	show starting, ending addresses of compiled code for source line <code>num</code>
<code>info source</code>	show name of current source file
<code>info sources</code>	list all source files in use
<code>forw regex</code>	search following source lines for <code>regex</code>
<code>rev regex</code>	search preceding source lines for <code>regex</code>

GDB under GNU Emacs

<code>M-x gdb</code>	run GDB under Emacs
<code>C-h m</code>	describe GDB mode
<code>M-s</code>	step one line (<code>step</code>)
<code>M-n</code>	next line (<code>next</code>)
<code>M-i</code>	step one instruction (<code>stepi</code>)
<code>C-c C-f</code>	finish current stack frame (<code>finish</code>)
<code>M-c</code>	continue (<code>cont</code>)
<code>M-u</code>	up <code>arg</code> frames (<code>up</code>)
<code>M-d</code>	down <code>arg</code> frames (<code>down</code>)
<code>C-x &</code>	copy number from point, insert at end (in source file)
<code>C-x SPC</code>	set break at point

GDB License

<code>show copying</code>	Display GNU General Public License
<code>show warranty</code>	There is NO WARRANTY for GDB.
	Display full no-warranty statement.

Copyright ©1991, '92, '93, '98 Free Software Foundation, Inc.
Roland H. Pesch

The author assumes no responsibility for any errors on this card.

This card may be freely distributed under the terms of the GNU General Public License.

Please contribute to development of this card by annotating it.
Improvements can be sent to bug-gdb@gnu.org.

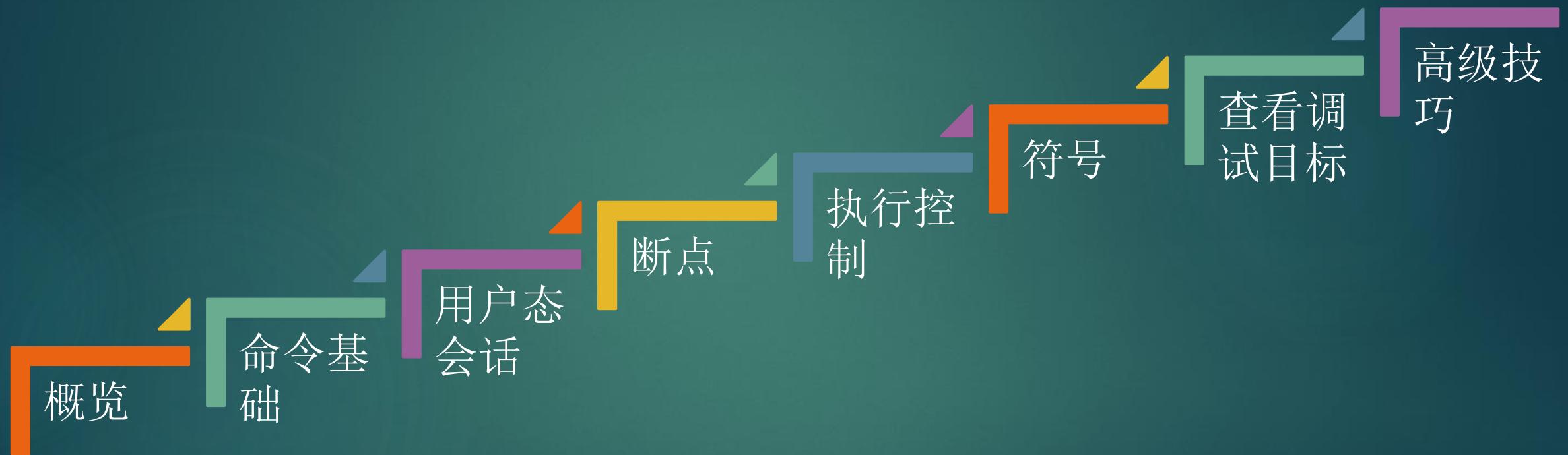
GDB itself is free software; you are welcome to distribute copies of it under the terms of the GNU General Public License. There is absolutely no warranty for GDB.

Get help

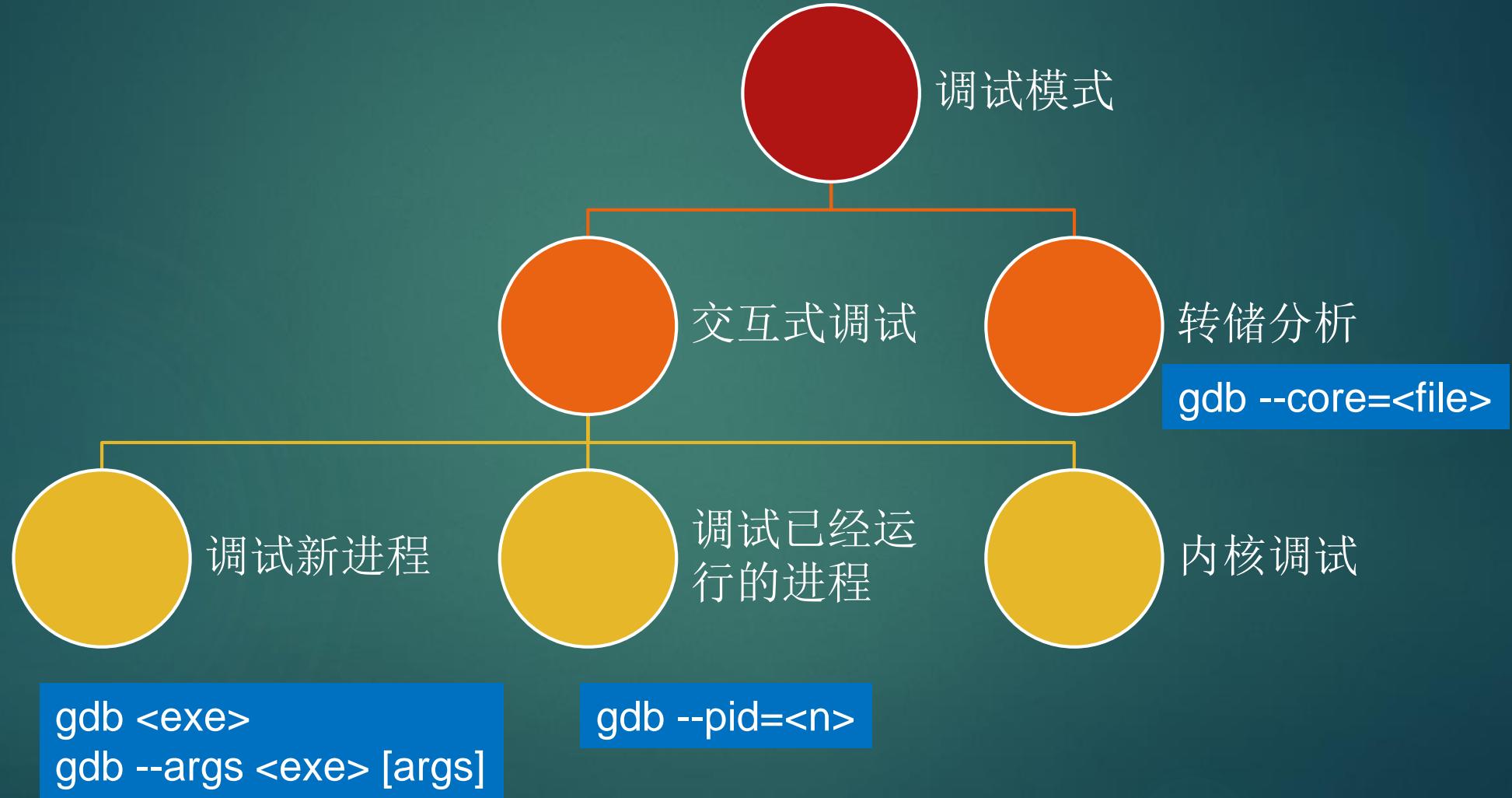
- ▶ help(h)
- ▶ help class
- ▶ apropos args
- ▶ complete args

```
(gdb) complete b
backtrace
bookmark
break
break-range
bt
(gdb) help complete
List the completions for the rest of the line as a command.
```

```
(gdb) apropos file
add-auto-load-safe-path -- Add entries to the list of directories from which it
is safe to auto-load files
add-inferior -- Add a new inferior
add-symbol-file -- Load symbols from FILE
append -- Append target code/data to a local file
append binary -- Append target code/data to a raw binary file
append binary memory -- Append contents of memory to a raw binary file
append binary value -- Append the value of an expression to a raw binary file
append memory -- Append contents of memory to a raw binary file
append value -- Append the value of an expression to a raw binary file
attach -- Attach to a process or file outside of GDB
break-range -- Set a breakpoint for an address range
compare-sections -- Compare section data on target to the exec file
compile file -- Evaluate a file containing source code
core-file -- Use FILE as core dump for examining memory and registers
detach -- Detach a process or file previously attached
directory -- Add directory DIR to beginning of search path for source files
disassemble -- Disassemble a specified section of memory
dump -- Dump target code/data to a local file
dump binary -- Write target code/data to a raw binary file
dump binary memory -- Write contents of memory to a raw binary file
dump binary value -- Write the value of an expression to a raw binary file
dump ihex -- Write target code/data to an intel hex file
```



调试模式



调试新进程

命令行指定exe

- `gdb <exe>`

命令行指定exe和参 数

- `gdb --args <exe> [args]`

使用命令

- `gdb`
- `file <exe>`
- `run [args]`

示例 1

- ▶ gdb
- ▶ file <exe>
- ▶ run [args]

```
raymond@unbuntu: ~/work/hdtrap
File Edit View Search Terminal Help
raymond@unbuntu:~$ cd work
raymond@unbuntu:~/work$ cd hdtrap
raymond@unbuntu:~/work/hdtrap$ gdb
GNU gdb (GDB) 7.2-ubuntu
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
(gdb) file ./hdtrap
Reading symbols from /home/raymond/work/hdtrap/hdtrap...done.
(gdb) run -args
Starting program: /home/raymond/work/hdtrap/hdtrap -args
Stuntman for xsw by Raymond (rev1.0)
running...

Program received signal SIGSEGV, Segmentation fault.
0x080484b8 in calc_md5 (data=0x8048590 "testing data-xxxxxxxx", nLen=20,
    md5=0x0) at md5.c:7
7          md5[0] = A;
(gdb)
```

示例2

- ▶ `gdb --args gcc -O2 -c foo.c`

args

- ▶ [~]\$ gdb --args pizzamaker --deep-dish --toppings=pepperoni
- ▶ ...
- ▶ (gdb) **show args**
- ▶ Argument list to give program being debugged when it is started is
- ▶ " --deep-dish --toppings=pepperoni".
- ▶ (gdb) b main
- ▶ Breakpoint 1 at 0x45467c: file oven.c, line 123.
- ▶ (gdb) run
- ▶ ...

设置和显示程序参数

- ▶ set args
 - ▶ Specify the arguments to be used the next time your program is run. If set args has no arguments, run executes your program with no arguments. Once you have run your program with arguments, using set args before the next run is the only way to run it again without arguments.
- ▶ show args
 - ▶ Show the arguments to give your program when it is started.

环境变量

- ▶ path directory
- ▶ show paths
- ▶ show environment [varname]
- ▶ set environment varname [=value]

```
(gdb) show path
Executable and object file path: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
(gdb) show environment
TERM=xterm
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd
=40;33;01:or=40;31;01:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=0
1;32:*.tar=01;31:*.tgz=01;31:*.arj=01;31:*.taz=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lz=01;31:*.xz=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31
```

工作目录

- ▶ 继承GDB的当前目录
- ▶ cd [directory]
 - ▶ Set the gdb working directory to directory. If not given, directory uses ~.
- ▶ pwd
 - ▶ Print the gdb working directory.

输入输出

- ▶ info terminal
- ▶ run > outfile
- ▶ tty /dev/ttyb

```
(gdb) info terminal
Inferior's terminal status (currently saved by GDB):
File descriptor flags = 0_RDWR
Process group = 3341
c_iflag = 0x6d02, c_oflag = 0x5,
c_cflag = 0x4bf, c_lflag = 0x8a3b
c_cc: 0x3 0x1c 0x7f 0x15 0x4 0x0 0x1 0xff 0x11 0x13 0x1a 0xff 0x12 0xf 0x17 0x10
      0xff 0x0 0x0
```

inferior

- ▶ n.部下，下属;次品;晚辈;[印]下角码 [ɪn'fɪrɪə(r)]
 - ▶ gdb represents the state of each program execution with an object called an inferior.
 - ▶ An inferior typically corresponds to a process, but is more general and applies also to targets that do not have processes.
 - ▶ Inferiors may be created before a process runs, and may be retained after a process exits.
- ▶ 下程

显示下程信息

```
(gdb) info inferiors
Num Description Executable
* 1 <null> /home/ge/work/llao3/baner
```

```
(gdb) info inferiors
Num Description Executable
7 <null> /home/ge/work/llao3/a.out
6 <null>
5 <null> /home/ge/work/llao3/baner
4 <null> /home/ge/work/llao3/baner
3 <null> /home/ge/work/llao3/baner
* 2 process 3341 /home/ge/work/llao3/baner
1 <null> /home/ge/work/llao3/baner
```

- ▶ 描述部分的null表示进程退出或者没有运行，或者已经detach

克隆下程

```
(gdb) clone-inferior -copies 2
Added inferior 2.
Added inferior 3.
(gdb) info inferiors
Num Description Executable
 3 <null>      /home/ge/work/llaoiao3/baner
 2 <null>      /home/ge/work/llaoiao3/baner
* 1 <null>      /home/ge/work/llaoiao3/baner
```

切换下程

- ▶ inferior infno
 - ▶ Make inferior number infno the current inferior. The argument infno is the inferior number assigned by gdb, as shown in the first field of the 'info inferiors' display.

```
(gdb) inferior 2
[Switching to inferior 2 [process 0] (/home/ge/work/llaoLao3/baner)]
(gdb) run
Starting program: /home/ge/work/llaoLao3/baner
baner <devno> r/w/s/i <value>
[Inferior 2 (process 2851) exited with code 0377]
```

增加下程

- ▶ add-inferior [-copies n] [-exec executable]

```
(gdb) add-inferior -copies 1 -exec a.out
Added inferior 7
Reading symbols from /home/ge/work/llaolao3/a.out...(no debugging symbols found)
...done.
(gdb) info inferiors
  Num  Description          Executable
    7  <null>                /home/ge/work/llaolao3/a.out
    6  <null>
    5  <null>                /home/ge/work/llaolao3/baner
    4  <null>                /home/ge/work/llaolao3/baner
    3  <null>                /home/ge/work/llaolao3/baner
*  2  <null>                /home/ge/work/llaolao3/baner
    1  <null>                /home/ge/work/llaolao3/baner
```

更多关于下程的命令

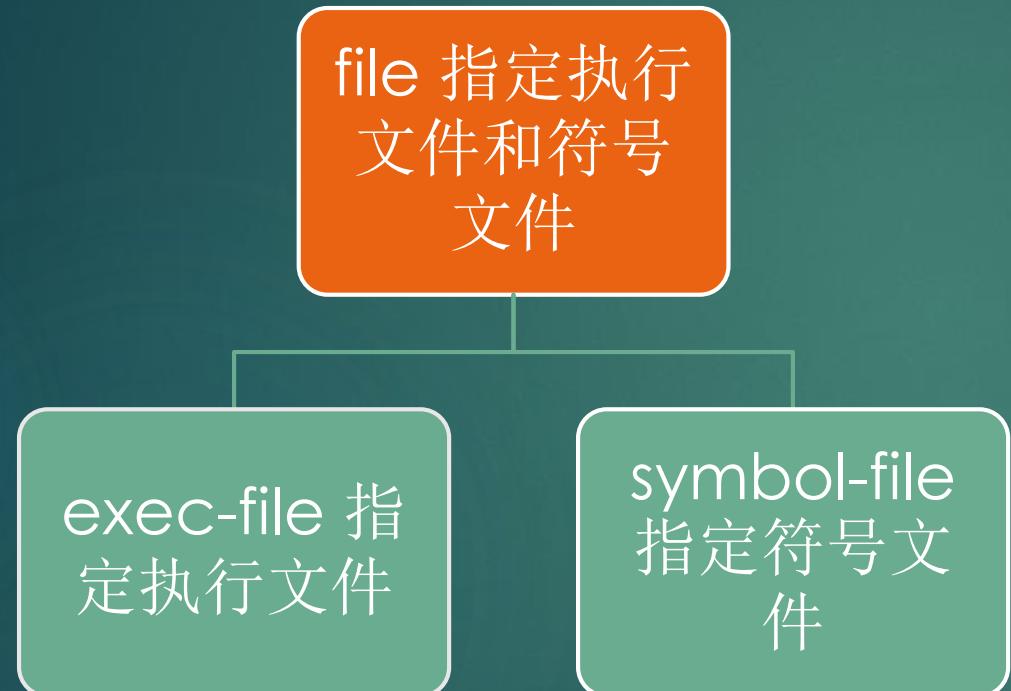
- ▶ remove-inferiors infno...
- ▶ detach inferior infno...
- ▶ kill inferiors infno...

file命令

- ▶ Use FILE as program to be debugged.
- ▶ It is read for its symbols, for getting the contents of pure memory, and it is the program executed when you use the `run' command.
- ▶ If FILE cannot be found as specified, your execution directory path (\$PATH) is searched for a command of that name.
- ▶ No arg means to have no executable file and no symbols.

```
(gdb) file notepad.exe
Reading symbols from notepad.exe...(no debugging symbols found)...done.
(gdb) r
Starting program: C:\windows\system32\notepad.exe
[New Thread 26604.0x529c]
[Inferior 1 (process 26604) exited normally]
```

文件命令



文件名作为参数，不带参数则清除此类信息

18 GDB Files

GDB needs to know the file name of the program to be debugged, both in order to read its symbol table and in order to start your program. To debug a core dump of a previous run, you must also tell GDB the name of the core dump file.

18.1 Commands to Specify Files

You may want to specify executable and core dump file names. The usual way to do this is at start-up time, using the arguments to GDB's start-up commands (see Chapter 2 [Getting In and Out of GDB], page 11).

开始运行

- ▶ run
- ▶ Start debugged program. You may specify arguments to give it.
- ▶ Args may include "*", or "[...]" ; they are expanded using "sh".
- ▶ Input and output redirection with ">", "<", or ">>" are also allowed.

- ▶ With no arguments, uses arguments last specified (with "run" or "set args").
- ▶ To cancel previous arguments and run with no arguments,
- ▶ use "set args" without arguments.

附加到已经运行的进程

命令行参数

- `gdb --pid=<n>`

使用attach命令

- `attach process-id`

终止调试会话

- ▶ 分离 - detach

```
(gdb) detach
Detaching from program: /home/ge/work/llaolao3/baner, process 3341
baner <devno> r/w/s/i <value>
(gdb) info inferiors
  Num  Description      Executable
    7  <null>          /home/ge/work/llaolao3/a.out
    6  <null>
    5  <null>          /home/ge/work/llaolao3/baner
    4  <null>          /home/ge/work/llaolao3/baner
    3  <null>          /home/ge/work/llaolao3/baner
*  2  <null>          /home/ge/work/llaolao3/baner
    1  <null>          /home/ge/work/llaolao3/baner
```

- ▶ 杀死 - quit

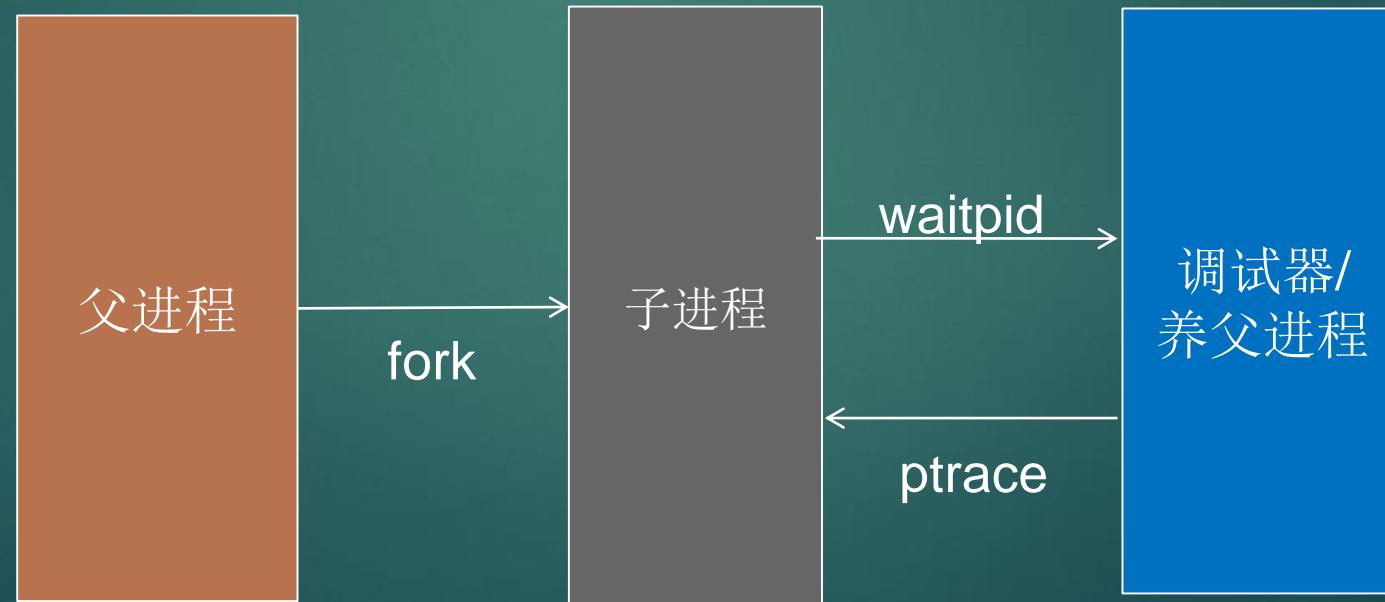
Quit gdb

- ▶ use the quit command (abbreviated q)
- ▶ type an end-of-file character (usually Ctrl-d)

进程跟踪

8

- ▶ Process Trace, Ptrace
- ▶ 最早实现在1979发布的Unix V7
- ▶ Unix/Linux用户态调试的主要依据

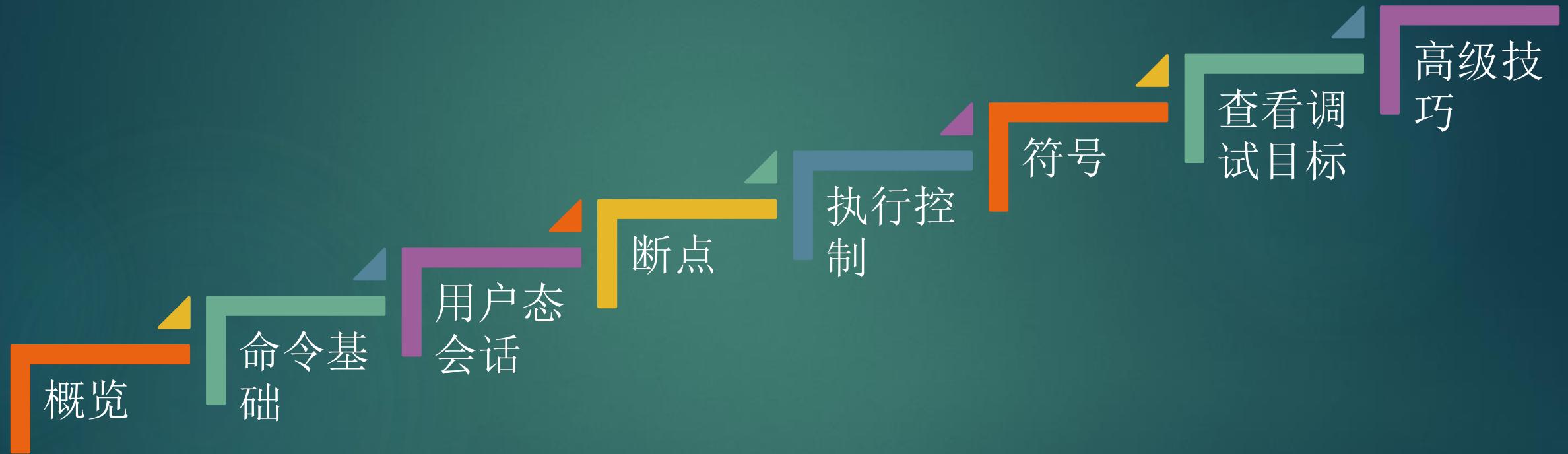


ptrace

- ▶ #include <sys/ptrace.h>
- ▶ long int ptrace(enum __ptrace_request request, pid_t pid, void * addr, void * data)
 - ▶ **PTRACE_ATTACH/ PTRACE_DETACH**
 - ▶ **PTRACE_PEEKTEXT, PTRACE_PEEKDATA,**
PTRACE_PEEKUSER
 - ▶ **PTRACE_POKETEXT, PTRACE_POKEDATA,**
PTRACE_POKEUSER
 - ▶ **PTRACE_SINGLESTEP**
 - ▶ **PTRACE_KILL**
- ▶ 系统调用

waitpid

- ▶ `#include <sys/types.h>`
- ▶ `#include <sys/wait.h>`
- ▶ `pid_t waitpid(pid_t pid, int *status, int options);`
- ▶ `WIFEXITED(status)`: 子进程正常退出
 - ▶ `WEXITSTATUS(status)`: 子进程的退出码
- ▶ `WIFSIGNALED(status)`: SIGINT, CTRL-C
 - ▶ `WTERMSIG(status)`: 终止信号编号
- ▶ `WIFSTOPPED(status)`: SIGSTOP, CTRL-Z
 - ▶ `WSTOPSIG(status)`: 停止信号编号



设置代码断点

软件
断点

`break` 普通

`tbreak` 一次性

`rbreak` 接受正则表达式成批设置

硬件
断点

`hbreak`

`thbreak` 一次性

两类断点

软件断点

- ▶ 基于CPU的断点指令，如x86的INT 3（机器码0xCC）
- ▶ 替换断点位置的指令
- ▶ CPU执行到此时触发断点异常
- ▶ 没有数量限制

硬件断点

- ▶ 基于CPU的调试寄存器，如x86的DR0 – DR7
- ▶ 不需要修改程序代码，可以针对EEPROM上的代码设置
- ▶ 有数量限制

指定位置(Location)

Linespec locations

行号

-/+偏移行

文件名: 行号

函数名

函数: 标号

文件名: 函数名

标号

Explicit locations

-source filename

-function function

-label label

-line number

Address locations

*address

示例 – 对函数名和地址设断点

```
(gdb) file baner
Reading symbols from /home/ge/work/llaolao3/baner...(no debugging symbols found)
...done.
(gdb) b main
Breakpoint 1 at 0x804876e
(gdb) info function usage
All functions matching regular expression "usage":

Non-debugging symbols:
0x080485e4  usage
(gdb) b *0x80485e4
Breakpoint 2 at 0x80485e4
(gdb) info b
Num      Type            Disp Enb Address     What
1        breakpoint      keep y    0x0804876e <main+4>
2        _breakpoint     keep y    0x080485e4 <usage>
```

示例 – 对源代码行设断点

```
(gdb) list usage
5      #include <errno.h>
6      #include <string.h>
7      #include "hdiocctl.h"
8
9      int usage()
10     {
11         printf("baner <devno> r/w/s/i <value>\n");
12         return -1;
13     }
14     void hd_ioctl(int fd, char * cmd, char *arg)
(gdb) b baner.c:11
Breakpoint 1 at 0x80485ea: file baner.c, line 11.
(gdb) b +2
Breakpoint 2 at 0x804860a: file baner.c, line 17.
(gdb) info b
Num      Type            Disp Enb Address      What
1        breakpoint      keep y  0x080485ea in usage at baner.c:11
2        breakpoint      keep y  0x0804860a in hd_ioctl at baner.c:17
```

- ▶ 第二个断点+2意为相对当前代码行之后 (after) 2行，上个list命令会影响当前代码行的位置
- ▶ For the list command, the current line is the last one printed; for the breakpoint commands, this is the line at which execution stopped

硬件断点

- ▶ VBOX虚拟机中设置失败

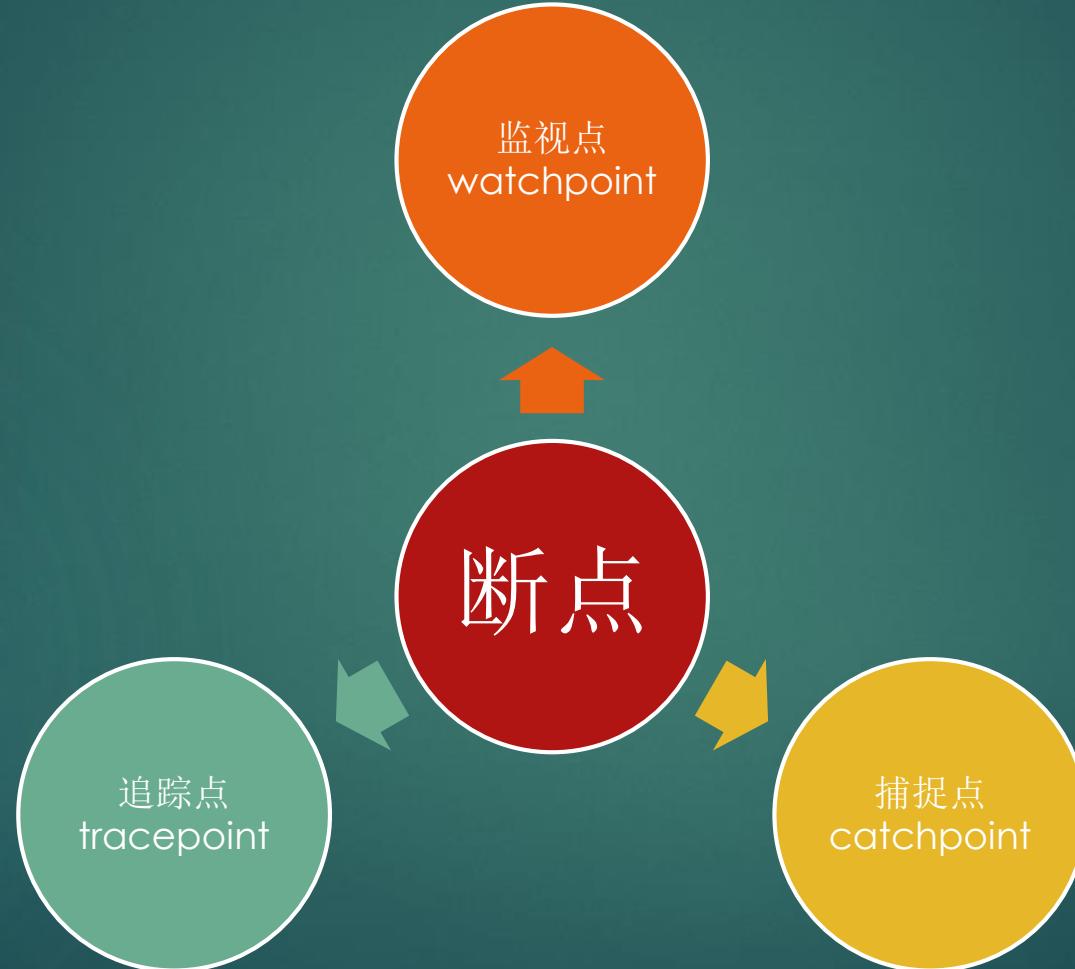
```
(gdb) hbreak hd_ioctl
No hardware breakpoint support in the target.
```

```
(gdb) info b
Num      Type            Disp Enb Address    What
1        hw breakpoint  keep y  <PENDING>  main
```

管理断点

- ▶ info b
- ▶ delete 1 #Delete breakpoint 1
- ▶ disable 1 # Disable the breakpoint 1
- ▶ enable 1 # Enable breakpoint 1
- ▶ delete #Delete all breakpoints
- ▶ clear sum #Clear any breakpoints at the entry to function sum

扩展断点



监视点

- ▶ 监视表达式，值变化时中断
 - ▶ The expression may be as simple as the value of a single variable, or as complex as many variables combined by operators.
 - ▶ watch a*b + c/d
 - ▶ watch *(int *)0x12345678
 - ▶ watch *global_ptr
- ▶ 数据断点 (data breakpoint)

监视点示例

实现

```
(gdb) info b
Num      Type            Disp Enb Address     What
1        breakpoint      keep y  0x0804877b in main at baner.c:53
        breakpoint already hit 1 time
2        hw watchpoint  keep y          fd
        breakpoint already hit 1 time
3        hw watchpoint  keep y          n
4        watchpoint     keep y         name
5        hw watchpoint  keep y         buffer[0]
6        hw watchpoint  keep y         buffer[10]
(gdb) watch buffer[5]
Hardware watchpoint 7: buffer[5]
(gdb) c
Continuing.
Warning:
Could not insert hardware watchpoint 6.
Could not insert hardware breakpoints:
You may have requested too many hardware breakpoints/watchpoints.
```

- ▶ 硬件平台相关
- ▶ X86上是使用硬件寄存器，数量有限制，最多4个

Depending on your system, watchpoints may be implemented in software or hardware. GDB does software watchpointing by single-stepping your program and testing the variable's value each time, which is hundreds of times slower than normal execution. (But this may still be worth it, to catch errors where you have no clue what part of your program is the culprit.)

On some systems, such as most PowerPC or x86-based targets, GDB includes support for hardware watchpoints, which do not slow down the running of your program.

自动删除

访问监视点

- ▶ rwatch [-l | -location] expr [thread thread-id] [mask maskvalue]
 - ▶ Set a watchpoint that will break when the value of expr is read by the program.
- ▶ awatch [-l | -location] expr [thread thread-id] [mask maskvalue]
 - ▶ Set a watchpoint that will break when expr is either read from or written into by the program.

```
(gdb) awatch fd
Hardware access (read/write) watchpoint 10: fd
(gdb) rwatch n
Hardware read watchpoint 11: n
(gdb) info b
Num      Type        Disp Enb Address   What
1        breakpoint  keep y  0x0804877b in main at baner.c:53
                                breakpoint already hit 1 time
10       acc watchpoint keep y          fd
11       read watchpoint keep y         n
(gdb) c
Continuing.
Hardware access (read/write) watchpoint 10: fd
Old value = 134513733
New value = 7
main (argc=4, argv=0xbfffff7f4) at baner.c:62
62           if(fd < 0)
(gdb) c
Continuing.
Hardware access (read/write) watchpoint 10: fd
Value = 7
0x080487d6 in main (argc=4, argv=0xbfffff7f4) at baner.c:62
62           if(fd < 0)
(gdb) c
Continuing.
Hardware access (read/write) watchpoint 10: fd
Value = 7
0x080488c4 in main (argc=4, argv=0xbfffff7f4) at baner.c:82
82           if((n = write(fd, argv[3], strlen(argv[3])))>0)
(gdb) c
Continuing.
Hardware read watchpoint 11: n
Value = 6
0x080488d5 in main (argc=4, argv=0xbfffff7f4) at baner.c:82
82           if((n = write(fd, argv[3], strlen(argv[3])))>0)
```

访问监视点示例

写fd触发

实际上是61行的写操作触发硬件断点，事后报告，导致gdb行号报告为62行

fd = open(name, O_RDWR);

读fd触发

读fd触发

读n做比较时触发，写时没有触发

线程约束

- ▶ `thread <threadno>`
- ▶ 进档指定线程遇到断点时才中断
- ▶ 适用于各类断点

`watch [-l|-location] expr [thread thread-id] [mask maskvalue]`

Set a watchpoint for an expression. GDB will break when the expression `expr` is written into by the program and its value changes. The simplest (and the most popular) use of this command is to watch the value of a single variable:

(gdb) `watch foo`

If the command includes a [`thread thread-id`] argument, GDB breaks only when the thread identified by `thread-id` changes the value of `expr`. If any other threads change the value of `expr`, GDB will not break. Note that watchpoints restricted to a single thread in this way only work with Hardware Watchpoints.

线程约束示例

- ▶ (gdb) b hd_ioctl thread 1
 - ▶ Breakpoint 10 at 0x8048603: file baner.c, line 16.

附加条件

- ▶ 直接使用if关键字附加在断点命令后
- ▶ 或者 condition bnum expression
- ▶ condition bnum 没有参数则删除之前设置的条件

条件断点示例

```
(gdb) b hd_ioctl thread 1 if fd>0
Unknown thread 1.
(gdb) run 0 i s 1
Starting program: /home/ge/work/llaolao3/baner 0 i s 1

Breakpoint 1, main (argc=5, argv=0xbfffff7f4) at baner.c:53
53      {
(gdb) b hd_ioctl thread 1 if fd>0
Breakpoint 12 at 0x8048603: file baner.c, line 16.
(gdb) info b
Num      Type            Disp Enb Address    What
1        breakpoint      keep y  0x0804877b in main at baner.c:53
                           breakpoint already hit 1 time
12       breakpoint      keep y  0x08048603 in hd_ioctl at baner.c:16 thread 1
                           stop only if fd>0
                           stop only in thread 1
(gdb) c
Continuing.

Breakpoint 12, hd_ioctl (fd=7, cmd=0xbfffff945 "s", arg=0xbfffff947 "1")
  at baner.c:16
16      _      int val = 0;
```

5.1.7 Breakpoint Command Lists

You can give any breakpoint (or watchpoint or catchpoint) a series of commands to execute when your program stops due to that breakpoint. For example, you might want to print the values of certain expressions, or enable other breakpoints.

`commands [range...]`

`... command-list ...`

`end`

Specify a list of commands for the given breakpoints. The commands themselves appear on the following lines. Type a line containing just `end` to terminate the commands.

To remove all commands from a breakpoint, type `commands` and follow it immediately with `end`; that is, give no commands.

With no argument, `commands` refers to the last breakpoint, watchpoint, or catchpoint set (not to the breakpoint most recently encountered). If the most recent breakpoints were set with a single command, then the `commands` will apply to all the breakpoints set by that command. This applies to breakpoints set by `rbreak`, and also applies when a single `break` command creates multiple breakpoints (see Section 10.2 [Ambiguous Expressions], page 116).

附件命令示例

```
(gdb) commands 12
Type commands for breakpoint(s) 12, one per line.
End with a line saying just "end".
>silent
>printf "fd is %d\n",fd
>continue
>end
```

```
(gdb) info b
Num      Type            Disp Enb Address     What
1        breakpoint      keep y  0x0804877b in main at baner.c:53
          breakpoint already hit 1 time
12       breakpoint     keep y   0x08048603 in hd_ioctl at baner.c:16 thread 1
          stop only if fd>0
          stop only in thread 1
          breakpoint already hit 1 time
          silent
          printf "fd is %d\n",fd
          continue
```

```
(gdb) c
Continuing.
fd is 7
ioctl cmd s value 1
[Inferior 1 (process 2640) exited normally]
(gdb) █
```

例2

```
(gdb) b do_mmap_pgoff
```

```
Breakpoint 1 at 0xffffffff8111a441: file mm/mmap.c, line 940.
```

```
(gdb) command 1
```

```
Type commands for when breakpoint 1 is hit, one per line.
```

```
End with a line saying just "end".
```

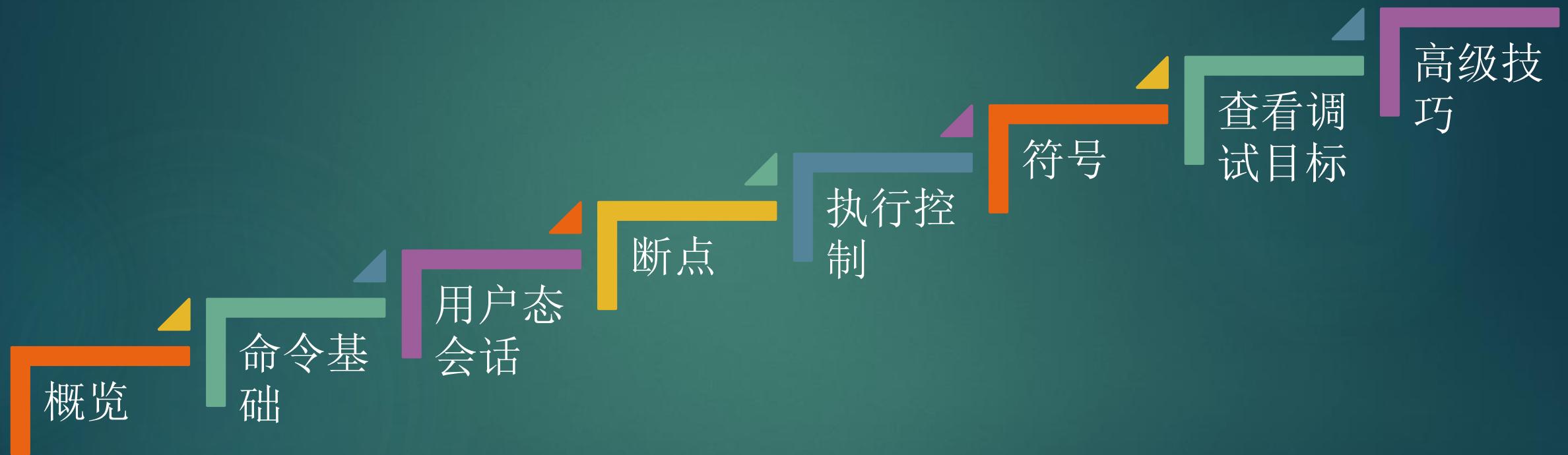
```
>print addr
```

```
>print len
```

```
>print prot
```

```
>end
```

```
(gdb)
```



单步跟踪

- ▶ stepi # Execute one instruction
- ▶ stepi 4 # Execute four instructions
- ▶ nexti # Like stepi, but proceed through function calls without stopping
- ▶ step # Execute one C statement

汇编级跟踪

- ▶ set disassemble-next-line on

```
(gdb) si
0x00000000004007c2      52          oldfunc(5,6);
  0x00000000004007bd <main+40>;       be 06 00 00 00  mov   $0x6,%esi
=> 0x00000000004007c2 <main+45>;       bf 05 00 00 00  mov   $0x5,%edi
  0x00000000004007c7 <main+50>;       e8 76 ff ff ff  callq 0x400742 <oldfunc>
(gdb)
0x00000000004007c7      52          oldfunc(5,6);
  0x00000000004007bd <main+40>;       be 06 00 00 00  mov   $0x6,%esi
  0x00000000004007c2 <main+45>;       bf 05 00 00 00  mov   $0x5,%edi
=> 0x00000000004007c7 <main+50>;       e8 76 ff ff ff  callq 0x400742 <oldfunc>
(gdb)
oldfunc (n=32767, m=-139936230) at binpatch.c:29
29  {
=> 0x0000000000400742 <oldfunc+0>;       55      push   %rbp
  0x0000000000400743 <oldfunc+1>;       48 89 e5    mov   %rsp,%rbp
  0x0000000000400746 <oldfunc+4>;       48 83 ec 20  sub   $0x20,%rsp
  0x000000000040074a <oldfunc+8>;       89 7d ec    mov   %edi,-0x14(%rbp)
  0x000000000040074d <oldfunc+11>;      89 75 e8    mov   %esi,-0x18(%rbp)
(gdb)
0x0000000000400743      29  {
  0x0000000000400742 <oldfunc+0>;       55      push   %rbp
=> 0x0000000000400743 <oldfunc+1>;       48 89 e5    mov   %rsp,%rbp
  0x0000000000400746 <oldfunc+4>;       48 83 ec 20  sub   $0x20,%rsp
  0x000000000040074a <oldfunc+8>;       89 7d ec    mov   %edi,-0x14(%rbp)
  0x000000000040074d <oldfunc+11>;      89 75 e8    mov   %esi,-0x18(%rbp)
(gdb)
```

汇编'窗口'

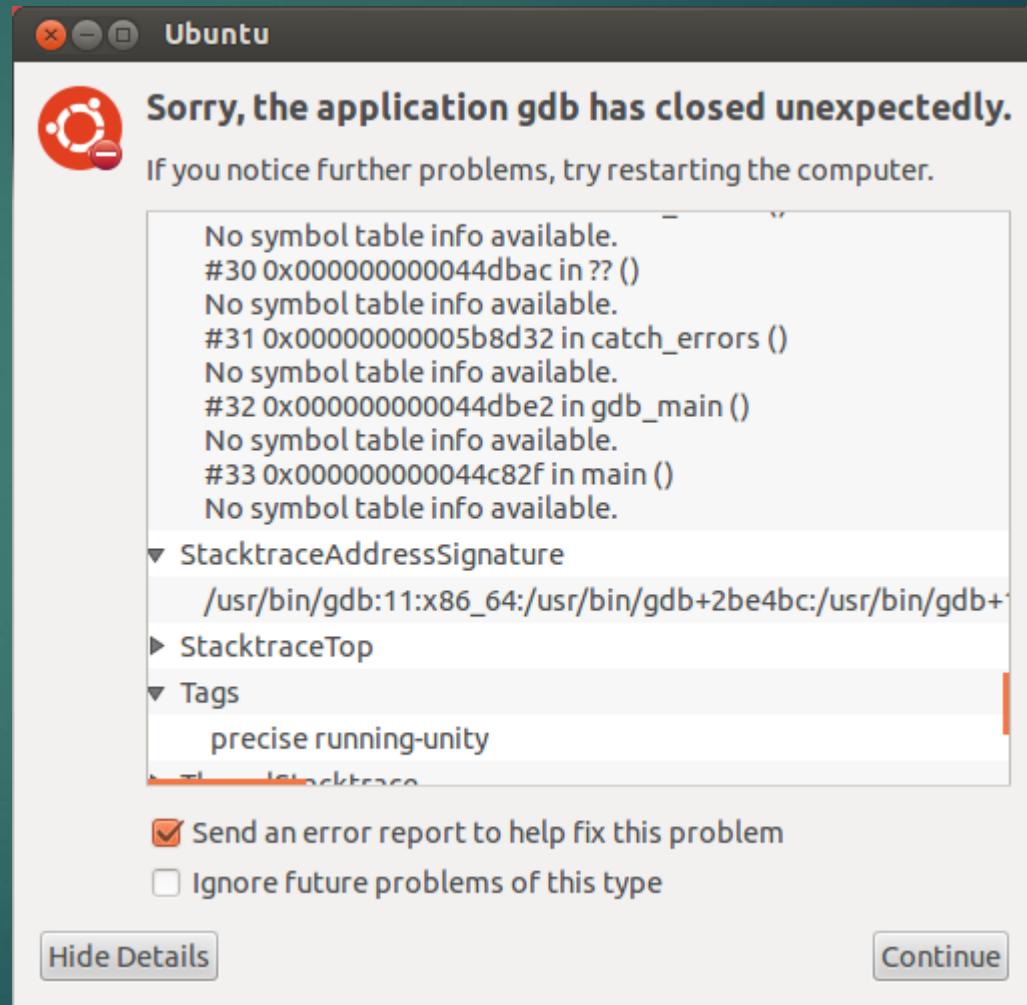
```
x - fed@fed-VirtualBox: ~/work/binpatch
B+ 0x40079d <main+8>      mov    %edi,-0x14(%rbp)
B+ 0x4007a0 <main+11>     mov    %rsi,-0x20(%rbp)
B+ 0x4007a4 <main+15>     movq   $0x0,-0x10(%rbp)
B+ 0x4007ac <main+23>     movl   $0x0,-0x4(%rbp)
B+ 0x4007b3 <main+30>     mov    $0x400b14,%edi
> 0x4007b8 <main+35>     callq  0x400590 <puts@plt>
B+ 0x4007bd <main+40>     mov    $0x6,%esi
B+ 0x4007c2 <main+45>     mov    $0x5,%edi
B+ 0x4007c7 <main+50>     callq  0x400742 <oldfunc>
B+ 0x4007cc <main+55>     mov    $0x4006f4,%esi
B+ 0x4007d1 <main+60>     mov    $0x400742,%edi

child process 2861 In: main                                         Line: 49   PC: 0x4007b8
(gdb) focus asm
Focus set to ASM window.
(gdb) ni
```

- ▶ (gdb)layout asm
- ▶ (gdb) focus asm
- ▶ (gdb) ni

Oops

- ▶ (gdb) show version
- ▶ GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2.1) 7.4-2012.04
- ▶ Copyright (C) 2012 Free Software Foundation, Inc.
- ▶ License GPLv3+: GNU GPL version 3 or later <<http://gnu.org/licenses/gpl.html>>
- ▶ This is free software: you are free to change and redistribute it.
- ▶ There is NO WARRANTY, to the extent permitted by law. Type "show copying"
- ▶ and "show warranty" for details.
- ▶ This GDB was configured as "x86_64-linux-gnu".



观察机器码

- ▶ echo 'asm("mov \$400835, %rax\n jmp *%rax\n");' > test.c
- ▶ gcc -c test.c
- ▶ objdump -d test.o
- ▶ display /3i \$pc

```
0000000000400742 <oldfunc>:  
400742: 55                      push  %rbp  
400743: 48 89 e5                mov   %rsp,%rbp  
400746: 48 83 ec 20             sub   $0x20,%rsp  
40074a: 89 7d ec                mov   %edi,-0x14(%rbp)  
40074d: 89 75 e8                mov   %esi,-0x18(%rbp)  
400750: c7 45 f4 08 00 00 00    movl  $0x8,-0xc(%rbp)  
400757: c7 45 f8 e7 03 00 00    movl  $0x3e7,-0x8(%rbp)  
40075e: 8b 45 f4                mov   -0xc(%rbp),%eax  
400761: 0f af 45 f8             imul  -0x8(%rbp),%eax  
400765: 0f af 45 e8             imul  -0x18(%rbp),%eax  
400769: 89 45 fc                mov   %eax,-0x4(%rbp)  
40076c: bf d3 0a 40 00             mov   $0x400ad3,%edi  
400771: e8 1a fe ff ff             callq 400590 <puts@plt>  
400776: b8 e8 0a 40 00             mov   $0x400ae8,%eax  
40077b: 8b 55 e8                mov   -0x18(%rbp),%edx  
40077e: 8b 4d ec                mov   -0x14(%rbp),%ecx  
400781: 89 ce                  mov   %ecx,%esi  
400783: 48 89 c7                mov   %rax,%rdi  
400786: b8 00 00 00 00             mov   $0x0,%eax  
40078b: e8 20 fe ff ff             callq 4005b0 <printf@plt>  
400790: 8b 45 fc                mov   -0x4(%rbp),%eax  
400793: c9                      leaveq  
400794: c3                      retq
```

恢复执行

- ▶ continue # Resume execution until the next breakpoint
- ▶ until 3 # Continue executing until program hits breakpoint 3
- ▶ finish # Resume execution until current function returns

调用函数

- ▶ call sum(1, 2) # Call sum(1,2) and print return value

```
(gdb) run 0 i s 1
Starting program: /home/ge/work/llaolao3/baner 0 i s 1
Breakpoint 1, main (argc=5, argv=0xbffff7f4) at baner.c:53
53      {
(gdb) call usage()
baner <devno> r/w/s/i <value>
$2 = -1
```

强制返回

- ▶ return expression
- ▶ You can cancel execution of a function call with the return command. If you give an expression argument, its value is used as the function's return value.

```
Breakpoint 1, Reading in symbols for libc-start.c...done.
main (argc=3, argv=0xbffff3f4) at baner.c:53
53      {
(gdb) return 1
Make main return now? (y or n) y
#0  __libc_start_main (main=0x804876a <main>,
    argc=3, ubp_av=0xbffff3f4, init=0x8048990 <__libc_csu_init>,
    fini=0x8048a00 <__libc_csu_fini>,
    rtld_fini=Reading in symbols for dl-fini.c...done.
0xb7fed230 <_dl_fini>, stack_end=0xbffff3ec) at libc-start.c:258
258      libc-start.c: No such file or directory.
(gdb)
Make __libc_start_main return now? (y or n) n
Not confirmed
(gdb) bt
#0  __libc_start_main (main=0x804876a <main>, argc=3, ubp_av=0xbffff3f4,
    init=0x8048990 <__libc_csu_init>, fini=0x8048a00 <__libc_csu_fini>,
    rtld_fini=0xb7fed230 <_dl_fini>, stack_end=0xbffff3ec) at libc-start.c:258
#1  0x08048551 in _start ()
```

触发中断

- ▶ `CTRL + C`

Suspending execution is done with the interrupt command when running in the background, or `Ctrl-c` during foreground execution.

反向单步

6 Running programs backward

When you are debugging a program, it is not unusual to realize that you have gone too far, and some event of interest has already happened. If the target environment supports it, GDB can allow you to “rewind” the program by running it backward.

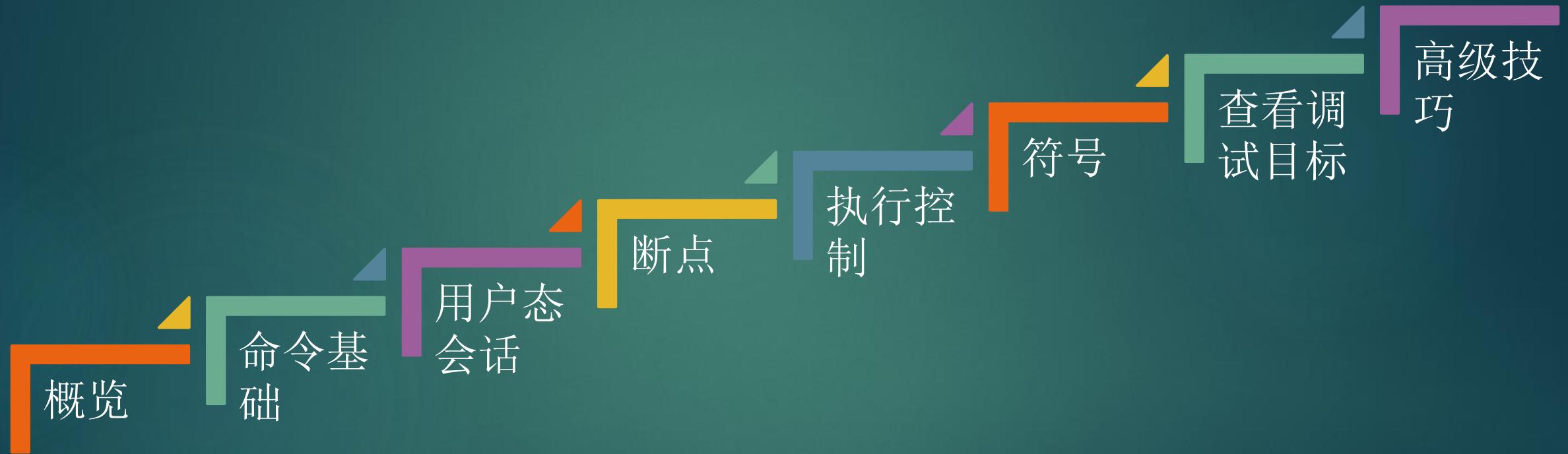
A target environment that supports reverse execution should be able to “undo” the changes in machine state that have taken place as the program was executing normally. Variables, registers etc. should revert to their previous values. Obviously this requires a great deal of sophistication on the part of the target environment; not all target environments can support reverse execution.

- ▶ # reverse-step, reverse-next, reverse-continue (rc)
- ▶ # target record
- ▶ # target record-full
- ▶ <http://stackoverflow.com/questions/1206872/go-to-previous-line-in-gdb>

利用RTIT 反向单步

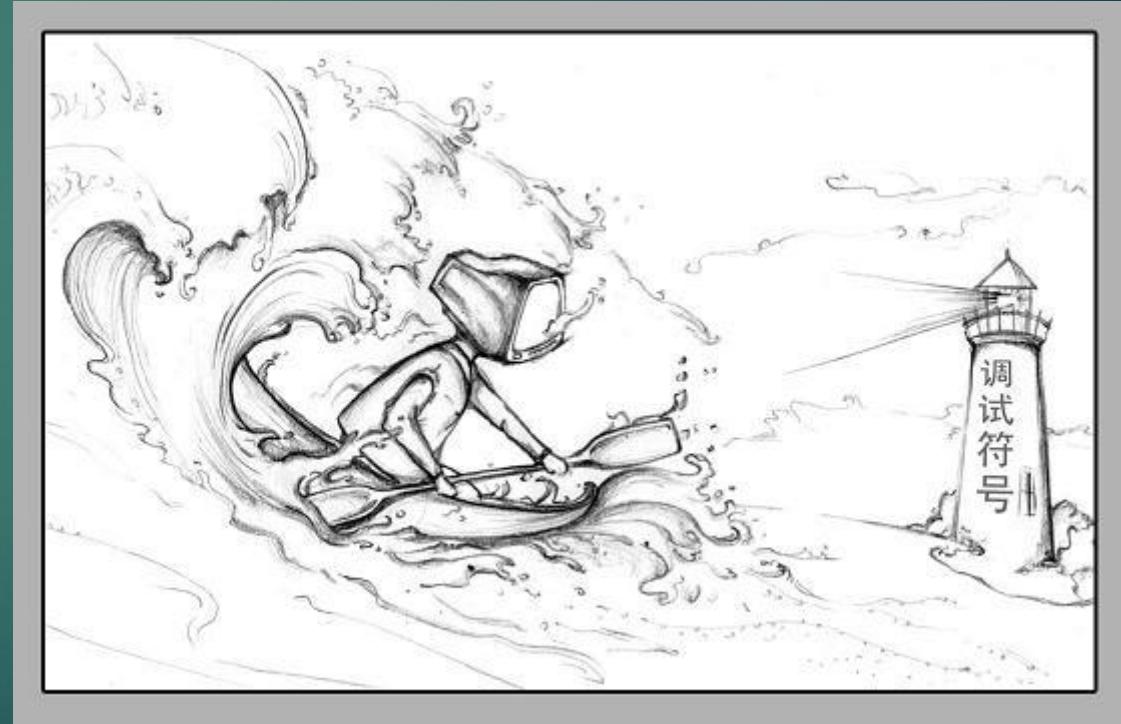
- ▶ Real time instruction trace, 又叫 intel processor trace
- ▶ Gdb 7.10 supports PT for “backwards debugging” (reverse-step)
- ▶ Uses perf interface, works as non root

```
(gdb) reverse-step
do_crash () at src/stack.c:59
59          return comp(&arg);
(gdb) record instruction-history -
67          0x0000000000400623 <do_crash+20>:    mov    -0x8(%rbp),%rax
68          0x0000000000400627 <do_crash+24>:    test   %rax,%rax
69          0x000000000040062a <do_crash+27>:    jne    0x400635
70          0x0000000000400635 <do_crash+38>:    mov    -0x8(%rbp),%rax
71          0x0000000000400639 <do_crash+42>:    mov    -0x8(%rbp),%rdx
72          0x000000000040063d <do_crash+46>:    mov    0x8(%rdx),%rdx
73          0x0000000000400641 <do_crash+50>:    add    (%rax),%rdx
74          0x0000000000400644 <do_crash+53>:    mov    %rdx,%rax
75          0x0000000000400647 <do_crash+56>:    leaveq 
76          0x0000000000400648 <do_crash+57>:    retq
```



调试符号

- ▶ 编译器对调试的重大贡献
 - ▶ 编译过程的副产品
- ▶ 衔接二进制程序与源程序的桥梁
- ▶ 对调试有着重要意义
- ▶ 源代码级调试必须
- ▶ 二进制跟踪时的灯塔



DWARF



- ▶ DWARF Debugging Information Format
- ▶ <http://www.dwarfstd.org>
- ▶ 功莫大焉

The screenshot shows the homepage of the DWARF Debugging Standard website at www.dwarfstd.org. The page features a header with the DWARF logo and the title "The DWARF Debugging Standard". A navigation bar below the header includes links for "HOME", "SPECIFICATIONS", "FAQ", and "ISSUES". The main content area is titled "Welcome to the DWARF Debugging Standard Website". It contains a brief description of DWARF as a debugging file format used for source level debugging across various languages and operating systems. It also mentions the availability of Version 4 of the standard for download and provides links for public comments and mailing lists.

HOME SPECIFICATIONS FAQ ISSUES

Welcome to the DWARF Debugging Standard Website

DWARF is a debugging file format used by many compilers and debuggers to support source level debugging. It addresses the requirements of a number of procedural languages, such as C, C++, and Fortran, and is designed to be extensible to other languages. DWARF is architecture independent and applicable to any processor or operating system. It is widely used on Unix, Linux and other operating systems, as well as in stand-alone environments.

Version 4 of the DWARF Debugging Information Format is available for download using the link below. See [Announcement](#) for additional details.

To submit a comment, please go to the "[Public Comment](#) page.

- [Download DWARF Debugging Standards](#)
- [Join the DWARF Discussion Mailing List](#)
- [DWARF Standards Committee Members](#)
- [Submission Standards](#)
- [Frequently Asked Questions](#)
- [DWARF Wiki](#)

The DWARF Version 4 Standard is available in either [PDF](#) or [MS Word](#) format.

再看ELF

```
ge@gewubox:~/work/llaolao3$ readelf -h baner
ELF Header:
  Magic: 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
        ELF32
  Class: ELF32
  Data: 2's complement, little endian
  Version: 1 (current)
  OS/ABI: UNIX - System V
  ABI Version: 0
  Type: EXEC (Executable file)
  Machine: Intel 80386
  Version: 0x1
  Entry point address: 0x8048530
  Start of program headers: 52 (bytes into file)
  Start of section headers: 5824 (bytes into file)
  Flags: 0x0
  Size of this header: 52 (bytes)
  Size of program headers: 32 (bytes)
  Number of program headers: 9
  Size of section headers: 40 (bytes)
  Number of section headers: 36
  Section header string table index: 33
```

段表

Section Headers:

[Nr]	Name	Type	Addr	Off	Size	ES	Flg	Lk	Inf	Al
[0]		NULL	00000000	000000	000000	00		0	0	0
[1]	.interp	PROGBITS	08048154	000154	000013	00	A	0	0	1
[2]	.note.ABI-tag	NOTE	08048168	000168	000020	00	A	0	0	4
[3]	.note.gnu.build-i	NOTE	08048188	000188	000024	00	A	0	0	4
[4]	.gnu.hash	GNU_HASH	080481ac	0001ac	000020	04	A	5	0	4
[5]	.dynsym	DYNSYM	080481cc	0001cc	0000f0	10	A	6	1	4
[6]	.dynstr	STRTAB	080482bc	0002bc	00009f	00	A	0	0	1
[7]	.gnu.version	VERSYM	0804835c	00035c	00001e	02	A	5	0	2
[8]	.gnu.version_r	VERNEED	0804837c	00037c	000030	00	A	6	1	4
[9]	.rel.dyn	REL	080483ac	0003ac	000008	08	A	5	0	4
[10]	.rel.plt	REL	080483b4	0003b4	000068	08	A	5	12	4
[11]	.init	PROGBITS	0804841c	00041c	00002e	00	AX	0	0	4
[12]	.plt	PROGBITS	08048450	000450	0000e0	04	AX	0	0	16
[13]	.text	PROGBITS	08048530	000530	00050c	00	AX	0	0	16
[14]	.fini	PROGBITS	08048a3c	000a3c	00001a	00	AX	0	0	4
[15]	.rodata	PROGBITS	08048a58	000a58	00013b	00	A	0	0	4
[16]	.eh_frame_hdr	PROGBITS	08048b94	000b94	000044	00	A	0	0	4
[17]	.eh_frame	PROGBITS	08048bd8	000bd8	000108	00	A	0	0	4
[18]	.ctors	PROGBITS	08049f14	000f14	000008	00	WA	0	0	4
[19]	.dtors	PROGBITS	08049f1c	000f1c	000008	00	WA	0	0	4
[20]	.jcr	PROGBITS	08049f24	000f24	000004	00	WA	0	0	4
[21]	.dynamic	DYNAMIC	08049f28	000f28	0000c8	08	WA	6	0	4
[22]	.got	PROGBITS	08049ff0	000ff0	000004	04	WA	0	0	4
[23]	.got.plt	PROGBITS	08049ff4	000ff4	000040	04	WA	0	0	4
[24]	.data	PROGBITS	0804a034	001034	000008	00	WA	0	0	4
[25]	.bss	NOBITS	0804a03c	00103c	000008	00	WA	0	0	4
[26]	.comment	PROGBITS	00000000	00103c	00002a	01	MS	0	0	1
[27]	.debug_aranges	PROGBITS	00000000	001066	000020	00		0	0	1
[28]	.debug_info	PROGBITS	00000000	001086	0001e8	00		0	0	1
[29]	.debug_abbrev	PROGBITS	00000000	00126e	000110	00		0	0	1
[30]	.debug_line	PROGBITS	00000000	00137e	000091	00		0	0	1
[31]	.debug_str	PROGBITS	00000000	00140f	0000bf	01	MS	0	0	1
[32]	.debug_loc	PROGBITS	00000000	0014ce	0000a8	00		0	0	1
[33]	.shstrtab	STRTAB	00000000	001576	000147	00		0	0	1
[34]	.symtab	SYMTAB	00000000	001c60	000530	10		35	51	4
[35]	.strtab	STRTAB	00000000	002190	0002ca	00		0	0	1

Key to Flags:

W (write), A (alloc), X (execute), M (merge), S (strings)

感受DWARF – 编译单元

Contents of the .debug_info section:

```
Compilation Unit @ offset 0x0:  
Length:      0x1e4 (32-bit)  
Version:     2  
Abbrev Offset: 0  
Pointer Size: 4  
<0><b>: Abbrev Number: 1 (DW_TAG_compile_unit)  
  <c>  DW_AT_producer    : (indirect string, offset: 0x38): GNU C 4.6.3  
  <10> DW_AT_language   : 1      (ANSI C)  
  <11> DW_AT_name       : (indirect string, offset: 0x0): baner.c  
  <15> DW_AT_comp_dir   : (indirect string, offset: 0xa8): /home/ge/work/llaolao3  
  <19> DW_AT_low_pc     : 0x80485e4  
  <1d> DW_AT_high_pc    : 0x804898f  
  <21> DW_AT_stmt_list  : 0x0  
<1><25>: Abbrev Number: 2 (DW_TAG_base_type)  
  <26> DW_AT_byte_size  : 1  
  <27> DW_AT_encoding   : 8      (unsigned char)  
  <28> DW_AT_name       : (indirect string, offset: 0x11): unsigned char
```

TAG(标签)

```
<1><a0>: Abbrev Number: 8 (DW_TAG_subprogram)
<a1>  DW_AT_external      : 1
<a2>  DW_AT_name          : (indirect string, offset: 0x8): hd_ioctl
<a6>  DW_AT_decl_file     : 1
<a7>  DW_AT_decl_line     : 14
<a8>  DW_AT_prototyped   : 1
<a9>  DW_AT_low_pc        : 0x80485fd
<ad>  DW_AT_high_pc       : 0x804876a
<b1>  DW_AT_frame_base   : 0x38 (location list)
<b5>  DW_AT_sibling       : <0x11d> |
<2><b9>: Abbrev Number: 9 (DW_TAG_formal_parameter)
<ba>  DW_AT_name          : fd
<bd>  DW_AT_decl_file     : 1
<be>  DW_AT_decl_line     : 14
<bf>  DW_AT_type          : <0x4f>
<c3>  DW_AT_location      : 2 byte block: 91 0  (DW_OP_fbreg: 0)
<2><c6>: Abbrev Number: 9 (DW_TAG_formal_parameter)
<c7>  DW_AT_name          : cmd
<cb>  DW_AT_decl_file     : 1
<cc>  DW_AT_decl_line     : 14
<cd>  DW_AT_type          : <0x6b>
<d1>  DW_AT_location      : 2 byte block: 91 4  (DW_OP_fbreg: 4)
<2><d4>: Abbrev Number: 9 (DW_TAG_formal_parameter)
<d5>  DW_AT_name          : arg
<d9>  DW_AT_decl_file     : 1
<da>  DW_AT_decl_line     : 14
<db>  DW_AT_type          : <0x6b>
<df>  DW_AT_location      : 2 byte block: 91 8  (DW_OP_fbreg: 8)
<2><e2>: Abbrev Number: 10 (DW_TAG_variable)
<e3>  DW_AT_name          : val
<e7>  DW_AT_decl_file     : 1
<e8>  DW_AT_decl_line     : 16
<e9>  DW_AT_type          : <0x4f>
<ed>  DW_AT_location      : 2 byte block: 91 6c  (DW_OP_fbreg: -20)
<2><f0>: Abbrev Number: 11 (DW_TAG_lexical_block)
<f1>  DW_AT_low_pc        : 0x8048628
<f5>  DW_AT_high_pc       : 0x804874d
```

产生符号

- ▶ \$ gcc -g -o baner baner.c
- ▶ -g --gen-debug **generate** debugging information

```
--debug-prefix-map OLD=NEW
                      map OLD to NEW in debug information
--defsym SYM=VAL      define symbol SYM to given value
--execstack            require executable stack for this object
--noexecstack          don't require executable stack for this object
--size-check=[error|warning]
                      ELF .size directive check (default --size-check=error)
-f
                      skip whitespace and comment preprocessing
-g --gen-debug         generate debugging information
--gstabs
--gstabs+              generate STABS debugging information
--gdwarf-2             generate DWARF2 debugging information
--hash-size=<value>    set the hash table size close to <value>
--help                 show this message and exit
```

Ubuntu的符号服务器

- ▶ <http://ddebs.ubuntu.com/pool/main/l/linux/>

ddebs.ubuntu.com/pool/main/l/linux/			
linux-cloud-tools-4.8.0-21-dbgsym_4.8.0-21.23_i386.ddeb	06-Oct-2016 06:53	832	
linux-image-3.2.0-25-generic-dbgsym_3.2.0-25.40_amd64.ddeb	24-May-2012 01:03	627M	
linux-image-3.2.0-25-generic-dbgsym_3.2.0-25.40_i386.ddeb	24-May-2012 02:08	635M	
linux-image-3.2.0-25-generic-pae-dbgsym_3.2.0-25.40_i386.ddeb	24-May-2012 02:32	635M	
linux-image-3.2.0-25-highbank-dbgsym_3.2.0-25.40_armhf.ddeb	24-May-2012 01:54	5.6M	
linux-image-3.2.0-25-omap-dbgsym_3.2.0-25.40_armel.ddeb	24-May-2012 00:33	288M	
linux-image-3.2.0-25-omap-dbgsym_3.2.0-25.40_armhf.ddeb	24-May-2012 01:47	288M	
linux-image-3.2.0-25-virtual-dbgsym_3.2.0-25.40_amd64.ddeb	24-May-2012 01:57	627M	
linux-image-3.2.0-25-virtual-dbgsym_3.2.0-25.40_i386.ddeb	24-May-2012 02:56	635M	
linux-image-3.2.0-26-generic-dbgsym_3.2.0-26.41_amd64.ddeb	14-Jun-2012 18:47	629M	
linux-image-3.2.0-26-generic-dbgsym_3.2.0-26.41_i386.ddeb	14-Jun-2012 17:45	637M	
linux-image-3.2.0-26-generic-pae-dbgsym_3.2.0-26.41_i386.ddeb	14-Jun-2012 17:58	637M	
linux-image-3.2.0-26-highbank-dbgsym_3.2.0-26.41_armhf.ddeb	14-Jun-2012 23:36	19M	
linux-image-3.2.0-26-omap-dbgsym_3.2.0-26.41_armel.ddeb	14-Jun-2012 22:15	290M	
linux-image-3.2.0-26-omap-dbgsym_3.2.0-26.41_armhf.ddeb	14-Jun-2012 23:26	290M	
linux-image-3.2.0-26-virtual-dbgsym_3.2.0-26.41_amd64.ddeb	14-Jun-2012 19:04	629M	
linux-image-3.2.0-26-virtual-dbgsym_3.2.0-26.41_i386.ddeb	14-Jun-2012 18:12	637M	
linux-image-3.2.0-27-generic-dbgsym_3.2.0-27.43_amd64.ddeb	06-Jul-2012 15:25	629M	

服务器的根目录

← → ⌂ ddebs.ubuntu.com

Index of /

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 clean.conf	01-Apr-2015 15:44	423	
 dbgsym-release-key.asc	04-Jul-2016 16:10	2.4K	
 dbgsym-release-key.asc.old	02-Sep-2008 18:28	1.8K	
 dists/	31-Jul-2016 12:14	-	
 pool/	19-Feb-2010 14:50	-	
 ubuntu/	17-Sep-2016 01:02	-	

Apache/2.2.22 (Ubuntu) Server at ddebs.ubuntu.com Port 80

下载内核符号文件



17



1. First create a `ddebs.list` using:

```
echo "deb http://ddebs.ubuntu.com $(lsb_release -cs) main restricted universe multiverse" | sudo tee /etc/apt/sources.list.d/ddebs.list
```



2. Then add the GPG key for `ddebs.ubuntu.com`:

```
wget -O - http://ddebs.ubuntu.com/dbgsym-release-key.asc | sudo apt-key add -
```

3. Then run:

```
sudo apt-get update
```

4. Then install the symbols package using:

```
sudo apt-get install linux-image-`uname -r`-dbgsym
```

This is rather huge (>680MB), so prepare for a wait while you download it.

示例2

```
fed@fed-VirtualBox:~/work$ sudo apt-get -c aptproxy.conf install linux-image-`uname -r`-dbgsym
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  linux-image-3.8.0-29-generic-dbgsym
0 upgraded, 1 newly installed, 0 to remove and 458 not upgraded.
Need to get 770 MB of archives.
After this operation, 2,532 MB of additional disk space will be used.
Get:1 http://ddebs.ubuntu.com/ precise-updates/main linux-image-3.8.0-29-generic-dbgsym amd64 3.8.0-29.42~precise1
[770 MB]
Fetched 770 MB in 1h 21min 54s (157 kB/s)
Selecting previously unselected package linux-image-3.8.0-29-generic-dbgsym.
(Reading database ... 142486 files and directories currently installed.)
Unpacking linux-image-3.8.0-29-generic-dbgsym (from .../linux-image-3.8.0-29-generic-dbgsym_3.8.0-29.42~precise1_amd64.ddeb) ...
Setting up linux-image-3.8.0-29-generic-dbgsym (3.8.0-29.42~precise1) ...
```

```
fed@fed-VirtualBox:~/work$ dpkg -S /lib/x86_64-linux-gnu/libc-2.15.so
libc6: /lib/x86_64-linux-gnu/libc-2.15.so
fed@fed-VirtualBox:~/work$ dpkg -S libc.so.6
libc6: /lib/x86_64-linux-gnu/libc.so.6
fed@fed-VirtualBox:~/work$ sudo apt-get -c aptproxy.conf install libc6-dbg
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  libc-bin libc-dev-bin libc6 libc6-dev
Suggested packages:
  glibc-doc
The following NEW packages will be installed:
  libc6-dbg
The following packages will be upgraded:
  libc-bin libc-dev-bin libc6 libc6-dev
4 upgraded, 1 newly installed, 0 to remove and 454 not upgraded.
Need to get 11.7 MB of archives.
After this operation, 19.9 MB of additional disk space will be used.
Do you want to continue [Y/n]? Y
Get:1 http://cn.archive.ubuntu.com/ubuntu/ precise-updates/main libc6-dev amd64 2.15-0ubuntu10.15 [2,943 kB]
Get:2 http://cn.archive.ubuntu.com/ubuntu/ precise-updates/main libc-dev-bin amd64 2.15-0ubuntu10.15 [84.7 kB]
Get:3 http://cn.archive.ubuntu.com/ubuntu/ precise-updates/main libc-bin amd64 2.15-0ubuntu10.15 [1,177 kB]
Get:4 http://cn.archive.ubuntu.com/ubuntu/ precise-updates/main libc6 amd64 2.15-0ubuntu10.15 [4,636 kB]
Get:5 http://cn.archive.ubuntu.com/ubuntu/ precise-updates/main libc6-dbg amd64 2.15-0ubuntu10.15 [2,888 kB]
Fetched 11.7 MB in 11s (1,055 kB/s)
Preconfiguring packages ...
(Reading database ... 146947 files and directories currently installed.)
Preparing to replace libc6-dev 2.15-0ubuntu10.4 (using .../libc6-dev_2.15-0ubuntu10.15_amd64.deb) ...
Unpacking replacement libc6-dev ...
Preparing to replace libc-dev-bin 2.15-0ubuntu10.4 (using .../libc-dev-bin_2.15-0ubuntu10.15_amd64.deb) ...
Unpacking replacement libc-dev-bin ...
Preparing to replace libc-bin 2.15-0ubuntu10.4 (using .../libc-bin_2.15-0ubuntu10.15_amd64.deb) ...
Unpacking replacement libc-bin ...
Processing triggers for man-db ...
Setting up libc-bin (2.15-0ubuntu10.15) ...
(Reading database ... 146946 files and directories currently installed.)
Preparing to replace libc6 2.15-0ubuntu10.4 (using .../libc6_2.15-0ubuntu10.15_amd64.deb) ...
Unpacking replacement libc6 ...
Setting up libc6 (2.15-0ubuntu10.15) ...
```

libc

学习用虚拟机-gebox

```
ge@gewubox:~$ sudo apt-get install libc6-dbg
[sudo] password for ge:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  libc-bin libc-dev-bin libc6 libc6-dev
Suggested packages:
  glibc-doc
The following NEW packages will be installed:
  libc6-dbg
The following packages will be upgraded:
  libc-bin libc-dev-bin libc6 libc6-dev
4 upgraded, 1 newly installed, 0 to remove and 357 not upgraded.
Need to get 12.8 MB of archives.
After this operation, 25.6 MB of additional disk space will be used.
Do you want to continue [Y/n]? Y
Get:1 http://cn.archive.ubuntu.com/ubuntu/ precise-updates/main libc6-dev i386 2
.15-0ubuntu10.15 [5,099 kB]
Get:2 http://cn.archive.ubuntu.com/ubuntu/ precise-updates/main libc-dev-bin i38
6 2.15-0ubuntu10.15 [77.7 kB]
Get:3 http://cn.archive.ubuntu.com/ubuntu/ precise-updates/main libc-bin i386 2.
```

加载符号文件

- ▶ file或者symbol-file [filename]
- ▶ Read symbol table information from file filename. PATH is searched when necessary.
- ▶ Use the file command to get both symbol table and program to run from the same file.

案例

```
(gdb) i shared
From          To            Syms Read  Shared Object Library
0xb7fde820  0xb7ff6baf  Yes (*)    /lib/ld-linux.so.2
0xb7e37f10  0xb7f6d44c  Yes (*)    /lib/i386-linux-gnu/libc.so.6
(*): Shared library is missing debugging information.
```

```
(gdb) inf shared
From          To            Syms Read  Shared Object Library
0xb7fde820  0xb7ff6b5f  Yes        /lib/ld-linux.so.2
0xb7e38f70  0xb7f6d35c  Yes        /lib/i386-linux-gnu/libc.so.6
```

符号文件信息

```
(gdb) info files
Symbols from "/home/ge/work/llaolao3/baner".
Unix child process:
    Using the running image of child process 3807.
    While running this, GDB does not access memory from...
Local exec file:
    `/home/ge/work/llaolao3/baner', file type elf32-i386.
Entry point: 0x8048530
0x08048154 - 0x08048167 @ 0x00000154 is .interp
0x08048168 - 0x08048188 @ 0x00000168 is .note.ABI-tag
0x08048188 - 0x080481ac @ 0x00000188 is .note.gnu.build-id
0x080481ac - 0x080481cc @ 0x000001ac is .gnu.hash
0x080481cc - 0x080482bc @ 0x000001cc is .dynsym
0x080482bc - 0x0804835b @ 0x000002bc is .dynstr
0x0804835c - 0x0804837a @ 0x0000035c is .gnu.version
0x0804837c - 0x080483ac @ 0x0000037c is .gnu.version_r
0x080483ac - 0x080483b4 @ 0x000003ac is .rel.dyn
0x080483b4 - 0x0804841c @ 0x000003b4 is .rel.plt
0x0804841c - 0x0804844a @ 0x0000041c is .init
0x08048450 - 0x08048530 @ 0x00000450 is .plt
0x08048530 - 0x08048a3c @ 0x00000530 is .text
0x08048a3c - 0x08048a56 @ 0x00000a3c is .fini
0x08048a58 - 0x08048b93 @ 0x00000a58 is .rodata
0x08048b94 - 0x08048bd8 @ 0x00000b94 is .eh_frame_hdr
0x08048bd8 - 0x08048ce0 @ 0x00000bd8 is .eh_frame
0x08049f14 - 0x08049f1c @ 0x00000f14 is .ctors
0x08049f1c - 0x08049f24 @ 0x00000f1c is .dtors
0x08049f24 - 0x08049f28 @ 0x00000f24 is .jcr
---Type <return> to continue, or q <return> to quit---
```

两阶段读取

- ▶ 对于大多数符号文件，GDB先是快速扫描符号文件，然后根据需要读取详细数据
- ▶ 可以使用`readnow`选项来强制读取所有符号信息

```
symbol-file [ -readnow ] filename  
file [ -readnow ] filename
```

You can override the GDB two-stage strategy for reading symbol tables by using the '`-readnow`' option with any of the commands that load symbol table information, if you want to be sure GDB has the entire symbol table available.

显示符号

- ▶ info variables regexp
- ▶ info classes regexp
- ▶ info functions regexp
- ▶ info types regexp

```
(gdb) info variables errno
All variables matching regular expression "errno":

File errno.c:
int rtld_errno;
int __libc_errno;
int errno;

File herrno.c:
int __libc_h_errno;
int h_errno;
```

查找函数

- ▶ info functions regexp

```
(gdb) info functions sprintf
All functions matching regular expression "sprintf":

File sprintf.c:
int __sprintf(char *, const char *, ...);

File asprintf.c:
int __asprintf(char **, const char *, ...);

File iovsprintf.c:
int __IO_vsprintf(char *, const char *, __gnuc_va_list);

File vasprintf.c:
int __IO_vasprintf(char **, const char *, __gnuc_va_list);

File sprintf_chk.c:
int __sprintf_chk(char *, int, size_t, const char *, ...);

File vsprintf_chk.c:
int __vsprintf_chk(char *, int, size_t, const char *, va_list);

File asprintf_chk.c:
int __asprintf_chk(char **, int, const char *, ...);

---Type <return> to continue, or q <return> to quit---
File vasprintf_chk.c:
int __GI__vasprintf_chk(char **, int, const char *, va_list);

Non-debugging symbols:
0x08048500  sprintf
0x08048500  sprintf@plt
```

符号命令归纳

- ▶ info address s #show where symbol s is stored
- ▶ info func [regex] #show names, types of defined functions (all, or matching regex)
- ▶ info var [regex] #show names, types of global variables (all, or matching regex)
- ▶ whatis [expr] #show data type of expr [or \$] without evaluating;
- ▶ptype [expr] #ptype gives more detail
- ▶ptype type #describe type, struct, union, or enum

符号和地址互查

`info address symbol`

Describe where the data for *symbol* is stored. For a register variable, this says which register it is kept in. For a non-register local variable, this prints the stack-frame offset at which the variable is always stored.

Note the contrast with ‘`print &symbol`’, which does not work at all for a register variable, and for a stack local variable prints the exact address of the current instantiation of the variable.

`info symbol addr`

Print the name of a symbol which is stored at the address *addr*. If no symbol is stored exactly at *addr*, GDB prints the nearest symbol and an offset from it:

```
(gdb) info symbol 0x54320
_initialize_vx + 396 in section .text
```

观察虚函数表

```
(gdb) x o  
0x602010:      0x0000000000400b50  
(gdb) x /2g $__  
0x400b50 <_ZTV6Object+16>:      0x000000000040097e      0x0000000000400996  
(gdb) info symbol $__  
Object::InsertOrder(int, void*) in section .text of /home/fed/work/vtable/vtable  
demo  
(gdb) info symbol 0x000000000040097e  
Object::Foo() in section .text of /home/fed/work/vtable/vtabledemo  
(gdb)
```

或

```
(gdb) x o  
0x602010:      0x0000000000400b50  
(gdb) x o  
0x602010:      0x0000000000400b50  
(gdb) x /2g 0x400b50  
0x400b50 <_ZTV6Object+16>:      0x000000000040097e      0x0000000000400996  
(gdb) info symbol ^CQuit  
(gdb) info symbol 0x000000000040097e  
Object::Foo() in section .text of /home/fed/work/vtable/vtabledemo  
(gdb) info symbol 0x0000000000400996  
Object::InsertOrder(int, void*) in section .text of /home/fed/work/vtable/vtable  
demo
```

查看源代码

dir names	add directory <i>names</i> to front of source path
dir	clear source path
show dir	show current source path
list	show next ten lines of source
list -	show previous ten lines
list <i>lines</i>	display source surrounding <i>lines</i> , specified as: [<i>file</i> :] <i>num</i> line number [in named file] [<i>file</i> :] <i>function</i> beginning of function [in named file] + <i>off</i> <i>off</i> lines after last printed - <i>off</i> <i>off</i> lines previous to last printed * <i>address</i> line containing <i>address</i>
list <i>f,l</i>	from line <i>f</i> to line <i>l</i>
info line <i>num</i>	show starting, ending addresses of compiled code for source line <i>num</i>
info source	show name of current source file
info sources	list all source files in use
forw <i>regex</i>	search following source lines for <i>regex</i>
rev <i>regex</i>	search preceding source lines for <i>regex</i>

源文件信息

```
(gdb) info source
Current source file is ioputs.c
Compilation directory is /build/eglibc-QV9xlv/eglibc-2.15/libio
Source language is c.
Compiled with DWARF 2 debugging format.
Does not include preprocessor macro info.
(gdb) info sources
Source files for which symbols have been read in:

/home/ge/work/llaolao3/baner.c, dl-runtime.c, ./libio/stdio.h,
./nptl/pthreadP.h, ./stdio-common/_itoa.h, ./sysdeps/gnu/_G_config.h,
./iconv/gconv.h, ./wcsmbcs/wchar.h, ./libio/libio.h,
./sysdeps/i386/dl-procinfo.c, ./sysdeps/i386/fpu_control.h,
./sysdeps/generic/ldsodefs.h, ./include/link.h,
./nptl/sysdeps/pthread/bits/libc-lock.h,
./nptl/sysdeps/unix/sysv/linux/internaltypes.h, ./nptl/descr.h,
./resolv/resolv.h, ./inet/netinet/in.h, ./bits/sockaddr.h,
./sysdeps/generic/unwind.h, ./nptl/./nptl_db/thread_db.h,
./nptl/sysdeps/pthread/pthread.h, ./nptl/sysdeps/pthread/list.h,
./sysdeps/i386/i686/hp-timing.h, ./sysdeps/i386/bits/setjmp.h,
./sysdeps/unix/sysv/linux/bits/sched.h, ./nptl/sysdeps/i386/i686/./tls.h,
./sysdeps/i386/bits/linkmap.h, ./elf/link.h, ./sysdeps/i386/bits/link.h,
./bits/elfclass.h, ./dlfcn/dlfcn.h,
./nptl/sysdeps/unix/sysv/linux/i386/bits/pthreadtypes.h,
./posix/sys/types.h, ./bits/types.h, ./elf/elf.h,
./sysdeps/generic/stdint.h, /usr/lib/gcc/i686-linux-gnu/4.6/include/stddef.h,
./sysdeps/i386/dl-irel.h, ./sysdeps/i386/dl-machine.h,
./sysdeps/i386/dl-trampoline.S, dl-debug.c, rtld.c, ./include/unistd.h,
./posix/unistd.h, ./include/assert.h, ./string/bits/string2.h,
./sysdeps/i386/dl-procinfo.h, ./sysdeps/i386/dl-tlsdesc.h, do-rel.h,
---Type <return> to continue, or q <return> to quit---
```

Install source files from Ubuntu

```
[~/src]$ apt-get source coreutils
[~/src]$ sudo apt-get install coreutils-dbgsym
[~/src]$ gdb /bin/ls
GNU gdb (GDB) 7.1-ubuntu
(gdb) list main
1192  ls.c: No such file or directory.
      in ls.c
(gdb) directory ~/src/coreutils-7.4/src/
Source directories searched: /home/nelhage/src/coreutils-7.4:$cdir:$cwd
(gdb) list main
1192  }
1193 }
```

安装libc源文件

- ▶ sudo apt-get source libc6-dev
- ▶ /home/ge/eglibc-2.15

```
ge@gewubox:~$ sudo apt-get source libc6-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
Picking 'eglibc' as source package instead of 'libc6-dev'
Need to get 25.4 MB of source archives.
Get:1 http://cn.archive.ubuntu.com/ubuntu/ precise-updates/main eglibc 2.15-0ubuntu10.15 (dsc) [5,829 B]
Get:2 http://cn.archive.ubuntu.com/ubuntu/ precise-updates/main eglibc 2.15-0ubuntu10.15 (tar) [23.5 MB]
Get:3 http://cn.archive.ubuntu.com/ubuntu/ precise-updates/main eglibc 2.15-0ubuntu10.15 (diff) [1,917 kB]
Fetched 25.4 MB in 24s (1,058 kB/s)
gpgv: Signature made Thu 26 May 2016 05:23:34 PM CST using RSA key ID 005E81F4
gpgv: Can't check signature: public key not found
dpkg-source: warning: failed to verify signature on ./eglibc_2.15-0ubuntu10.15.dsc
dpkg-source: info: extracting eglibc in eglibc-2.15
dpkg-source: info: unpacking eglibc_2.15.orig.tar.gz
dpkg-source: info: applying eglibc_2.15-0ubuntu10.15.diff.gz
```

设置搜索路径

```
directory dirname ...
dir dirname ...
```

- ▶ Add directory dirname to the front of the source path. Several directory names may be given to this command, separated by ‘:’ (‘;’ on MS-DOS and MSWindows, where ‘:’ usually appears as part of absolute file names) or whitespace.
- ▶ ‘\$cdir’ to refer to the compilation directory
- ▶ ‘\$cwd’ to refer to the current working directory

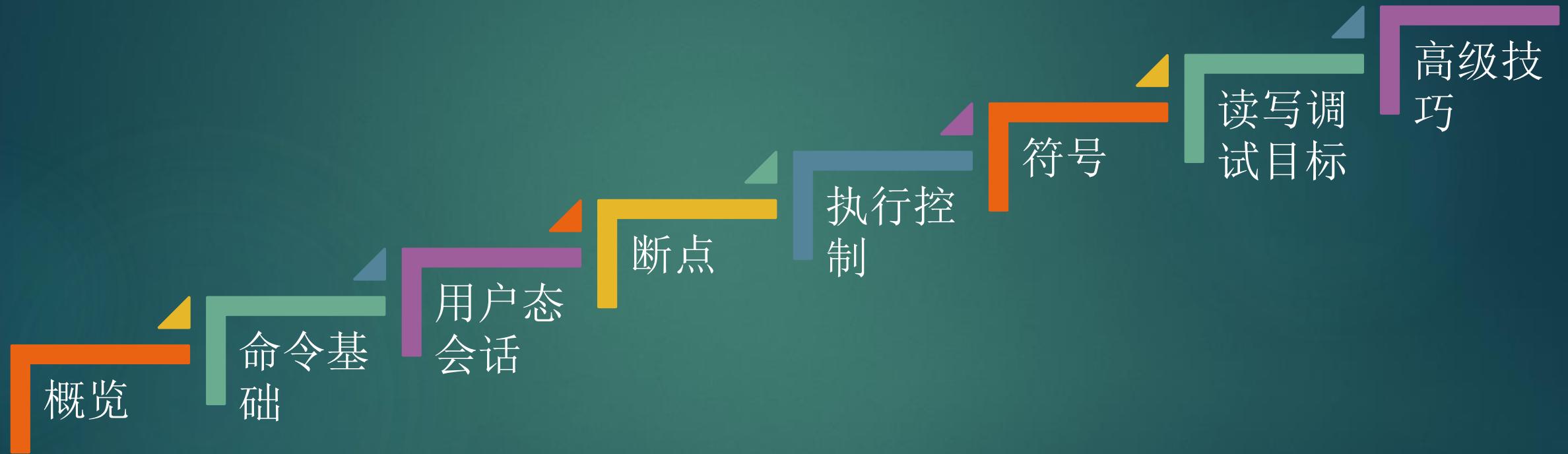
```
(gdb) directory /home/ge/eglibc-2.15/libio
Source directories searched: /home/ge/eglibc-2.15/libio:/home/ge/eglibc-2.15:/home/ge/eglibc:$cdir:$cwd
(gdb) list
30      #include <limits.h>
31
32      int
33      _IO_puts (str)
34          const char *str;
35      {
36          int result = EOF;
37          _IO_size_t len = strlen (str);
38          _IO_acquire_lock (_IO_stdout);
39      }
```

安装内核源文件

```
[~/src]$ apt-get source linux-image-2.6.32-25-generic
[~/src]$ sudo apt-get install linux-image-2.6.32-25-generic-dbgsym
[~/src]$ gdb /usr/lib/debug/boot/vmlinux-2.6.32-25-generic
(gdb) list schedule
5519  /build/buildd/linux-2.6.32/kernel/sched.c: No such file or directory.
      in /build/buildd/linux-2.6.32/kernel/sched.c
(gdb) set substitute-path /build/buildd/linux-2.6.32
/home/nelhage/src/linux-2.6.32/
(gdb) list schedule
5519
5520 static void put_prev_task(struct rq *rq, struct task_struct *p)
5521 {
```

常用命令

```
ge@gewubox:~$ apt-get source coreutils
Reading package lists... Done
Building dependency tree
Reading state information... Done
Need to get 12.6 MB of source archives.
Get:1 http://cn.archive.ubuntu.com/ubuntu/ precise-updates/main coreutils 8.13-3
ubuntu3.3 (dsc) [1,650 B]
Get:2 http://cn.archive.ubuntu.com/ubuntu/ precise-updates/main coreutils 8.13-3
ubuntu3.3 (tar) [11.7 MB]
Get:3 http://cn.archive.ubuntu.com/ubuntu/ precise-updates/main coreutils 8.13-3
ubuntu3.3 (diff) [930 kB]
Fetched 12.6 MB in 11s (1,114 kB/s)
gpgv: Signature made Wed 14 Jan 2015 11:32:42 AM CST using RSA key ID 9D8D2E97
gpgv: Can't check signature: public key not found
dpkg-source: warning: failed to verify signature on ./coreutils_8.13-3ubuntu3.3.
dsc
dpkg-source: info: extracting coreutils in coreutils-8.13
dpkg-source: info: unpacking coreutils_8.13.orig.tar.gz
dpkg-source: info: applying coreutils_8.13-3ubuntu3.3.diff.gz
ge@gewubox:~$ ls
coreutils-8.13          Desktop    examples.desktop  sf.sh
coreutils_8.13-3ubuntu3.3.diff.gz dev.txt    Music           Templates
coreutils_8.13-3ubuntu3.3.dsc Documents  Pictures        Videos
coreutils_8.13.orig.tar.gz   Downloads Public         work
```



观察寄存器

```
(gdb) info reg
eax            0xbffff7f4      - 1073743884
ecx            0xbffff7f4      - 1073743884
edx            0xbffff784      - 1073743996
ebx            0xb7fc6ff4      - 1208193036
esp            0xbffff6f0      0xbffff6f0
ebp            0xbffff758      0xbffff758
esi             0x0          0
edi             0x0          0
eip             0x804877b      0x804877b <main+17>
eflags          0x286      [ PF SF IF ]
cs              0x73        115
ss              0x7b        123
ds              0x7b        123
es              0x7b        123
fs              0x0          0
gs              0x33        51
```

修改寄存器

- ▶ (gdb)set \$<name>=<value>

```
(gdb) info reg r14  
r14          0x0      0  
(gdb) set $r14=1  
(gdb) info reg r14  
r14          0x1      1  
(gdb) █
```

栈

<code>backtrace [n]</code>	print trace of all frames in stack; or of n frames—innermost if $n > 0$, outermost if $n < 0$
<code>bt [n]</code>	
<code>frame [n]</code>	select frame number n or frame at address n ; if no n , display current frame
<code>up n</code>	select frame n frames up
<code>down n</code>	select frame n frames down
<code>info frame [addr]</code>	describe selected frame, or frame at $addr$
<code>info args</code>	arguments of selected frame
<code>info locals</code>	local variables of selected frame
<code>info reg [rn]...</code>	register values [for regs rn] in selected frame; <code>all-reg</code> includes floating point
<code>info all-reg [rn]</code>	

栈回溯

▶ backtrace

```
(gdb) bt
#0  hd_ioctl (fd=7, cmd=0xbffff945 "s", arg=0xbffff947 "1") at baner.c:16
#1  0x08048967 in main (argc=5, argv=0xbffff7f4) at baner.c:96
```

参数和局部变量

```
(gdb) where
#0  hd_ioctl (fd=7, cmd=0xbffff945 "s", arg=0xbffff947 "1") at baner.c:16
#1  0x08048967 in main (argc=5, argv=0xbffff7f4) at baner.c:96
(gdb) info locals
val = -1073743551
(gdb) info local
val = -1073743551
(gdb) info args
fd = 7
cmd = 0xbffff945 "s"
arg = 0xbffff947 "1"
```

切换栈帧

► frame <no>

```
Breakpoint 1, Reading in symbols for libc-start.c...done.
main (argc=3, argv=0xbffff3f4) at baner.c:53
53      {
(gdb) b _IO_puts
Reading in symbols for ioputs.c...done.
Breakpoint 4 at 0xb7e88830: file ioputs.c, line 35.
(gdb) c
Continuing.

Breakpoint 2, __printf (format=0x8048b03 "open file %s failed with %d\n")
    at printf.c:30
30      printf.c: No such file or directory.
(gdb) bt
#0  __printf (format=0x8048b03 "open file %s failed with %d\n") at printf.c:30
#1  0x080487f8 in main (argc=3, argv=0xbffff3f4) at baner.c:64
(gdb) info locals
arg = <optimized out>
done = 134515459
(gdb) frame 1
#1  0x080487f8 in main (argc=3, argv=0xbffff3f4) at baner.c:64
64          printf("open file %s failed with %d\n", name, errno);
(gdb) info locals
n = -1208192028
fd = -1
name = "/dev/huadengw\000\377\377\326", <incomplete sequence \345\267>
buffer = "\364o\374\267eN\345\267\060\322\376\267\000\000\000\231\211\004\b"
```

观察变量和内存

```
print [/f] [expr]    show value of expr [or last value $]
p [/f] [expr]        according to format f:
    x               hexadecimal
    d               signed decimal
    u               unsigned decimal
    o               octal
    t               binary
    a               address, absolute and relative
    c               character
    f               floating point

x [/Nuf] expr       examine memory at address expr; optional
                    format spec follows slash
    N               count of how many units to display
    u               unit size; one of
        b             individual bytes
        h             halfwords (two bytes)
        w             words (four bytes)
        g             giant words (eight bytes)
    f               printing format. Any print format, or
        s             null-terminated string
        i             machine instructions
```

X

```
[~]$ grep saved_command /proc/kallsyms  
ffffffff81946000 B saved_command_line
```

```
(gdb) x/s 0xffffffff81946000  
ffffffff81946000 <>: "root=/dev/sda1 quiet"
```

示例

```
(gdb) x /32bx arg
0xbffff947: 0x31 0x00 0x53 0x48 0x45 0x4c 0x4c 0x3d
0xbffff94f: 0x2f 0x62 0x69 0x6e 0x2f 0x62 0x61 0x73
0xbffff957: 0x68 0x00 0x54 0x45 0x52 0x4d 0x3d 0x78
0xbffff95f: 0x74 0x65 0x72 0x6d 0x00 0x55 0x53 0x45
```

```
(gdb) p arg[0]
$6 = 49 '1'
(gdb) p arg[1]
$7 = 0_ '\000'
```



- ▶ (gdb) p *&a[0]@10
\$1 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
- ▶ to view elements 550-553 in a long vector, all you do is
- ▶ 'p *&a[550]@4'

```
(gdb) p argv[0]@6
$14 = {0xbfffff924 "/home/ge/work/llaolao3/baner", 0xbfffff941 "0",
        0xbfffff943 "i", 0xbfffff945 "s", 0xbfffff947 "1", 0x0}
```

修改内存

- ▶ (gdb) set <var>=<exp>

```
(gdb) set worker=2
(gdb) info locals
worker = 2
n = 0
(gdb) █
```

(gdb) help set variable

Evaluate expression EXP and assign result to variable VAR, using assignment syntax appropriate for the current language (VAR = EXP or VAR := EXP for example). VAR may be a debugger "convenience" variable (names starting with \$), a register (a few standard names starting with \$), or an actual variable in the program being debugged. EXP is any valid expression. This may usually be abbreviated to simply "set".

反汇编

disassem [addr] display memory as machine instructions

```
(gdb) disas main
Dump of assembler code for function main:
0x0804876a <+0>: push %ebp
0x0804876b <+1>: mov %esp,%ebp
0x0804876d <+3>: push %edi
0x0804876e <+4>: and $0xffffffff,%esp
0x08048771 <+7>: sub $0x60,%esp
0x08048774 <+10>: mov 0xc(%ebp),%eax
0x08048777 <+13>: mov %eax,0x1c(%esp)
0x0804877b <+17>: mov %gs:0x14,%eax
0x08048781 <+23>: mov %eax,0x5c(%esp)
0x08048785 <+27>: xor %eax,%eax
0x08048787 <+29>: cmpl $0x2,0x8(%ebp)
0x0804878b <+33>: jg 0x8048797 <main+45>
0x0804878d <+35>: call 0x80485e4 <usage>
0x08048792 <+40>: jmp 0x8048978 <main+526>
0x08048797 <+45>: mov 0x1c(%esp),%eax
0x0804879b <+49>: add $0x4,%eax
0x0804879e <+52>: mov (%eax),%edx
0x080487a0 <+54>: mov $0x8048af4,%eax
0x080487a5 <+59>: mov %edx,0x8(%esp)
0x080487a9 <+63>: mov %eax,0x4(%esp)
0x080487ad <+67>: lea 0x34(%esp),%eax
0x080487b1 <+71>: mov %eax,(%esp)
---Type <return> to continue, or q <return> to quit---
```

x/i

(gdb) x/5i schedule

```
0xffffffff8154804a <schedule>: push %rbp  
0xffffffff8154804b <schedule+1>: mov $0x11ac0,%rdx  
0xffffffff81548052 <schedule+8>: mov %gs:0xb588,%rax  
0xffffffff8154805b <schedule+17>: mov %rsp,%rbp  
0xffffffff8154805e <schedule+20>: push %r15
```

x/10i sum Examine first 10 instructions of function sum

https://blogs.oracle.com/ksplice/entry/8_gdb_tricks_you_should

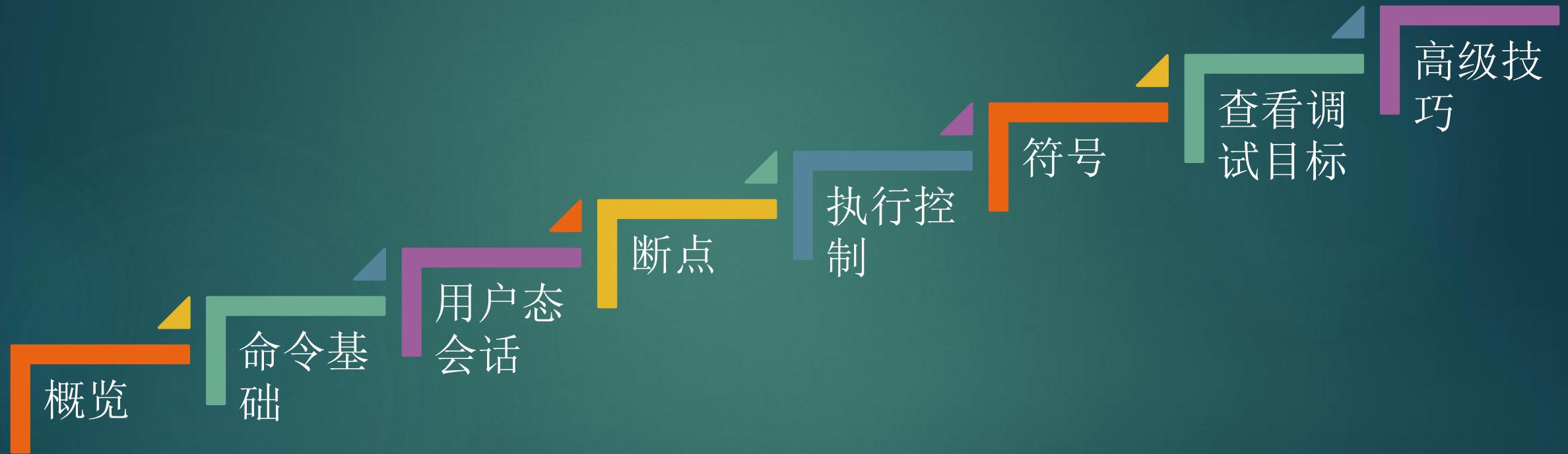
X86汇编语言

INTEL语法

- 先是目标，然后是源，也就是从右向左赋值
- Windows上流行

AT&T语法

- 先是源，然后是目标，也就是从左向右赋值
- Unix和Linux上流行



信号

Signal	Stop	Print	Pass to program	Description
SIGHUP	Yes	Yes	Yes	Hangup
SIGINT	Yes	Yes	No	Interrupt
SIGQUIT	Yes	Yes	Yes	Quit
SIGILL	Yes	Yes	Yes	Illegal instruction
SIGTRAP	Yes	Yes	No	Trace/breakpoint trap
SIGABRT	Yes	Yes	Yes	Aborted
SIGEMT	Yes	Yes	Yes	Emulation trap
SIGFPE	Yes	Yes	Yes	Arithmetic exception
SIGKILL	Yes	Yes	Yes	Killed
SIGBUS	Yes	Yes	Yes	Bus error
SIGSEGV	Yes	Yes	Yes	Segmentation fault
SIGSYS	Yes	Yes	Yes	Bad system call
SIGPIPE	Yes	Yes	Yes	Broken pipe
SIGALRM	No	No	Yes	Alarm clock
SIGTERM	Yes	Yes	Yes	Terminated
SIGURG	No	No	Yes	Urgent I/O condition
SIGSTOP	Yes	Yes	Yes	Stopped (signal)
SIGTSTP	Yes	Yes	Yes	Stopped (user)
SIGCONT	Yes	Yes	Yes	Continued
SIGCHLD	No	No	Yes	Child status changed
SIGTTIN	Yes	Yes	Yes	Stopped (tty input)

处理规则

```
(gdb) info handle
Signal      Stop     Print    Pass to program Description
SIGHUP      Yes      Yes      Yes      Hangup
SIGINT      Yes      Yes      No       Interrupt
SIGQUIT     Yes      Yes      Yes      Quit
SIGILL      Yes      Yes      Yes      Illegal instruction
SIGTRAP     Yes      Yes      No       Trace/breakpoint trap
SIGABRT     Yes      Yes      Yes      Aborted
SIGEMT      Yes      Yes      Yes      Emulation trap
SIGFPE      Yes      Yes      Yes      Arithmetic exception
SIGKILL     Yes      Yes      Yes      Killed
SIGBUS      Yes      Yes      Yes      Bus error
SIGSEGV     Yes      Yes      Yes      Segmentation fault
SIGSYS      Yes      Yes      Yes      Bad system call
SIGPIPE     Yes      Yes      Yes      Broken pipe
SIGALRM    No       No       Yes      Alarm clock
SIGTERM     Yes      Yes      Yes      Terminated
SIGURG      No       No       Yes      Urgent I/O condition
SIGSTOP     Yes      Yes      Yes      Stopped (signal)
SIGTSTP    Yes      Yes      Yes      Stopped (user)
SIGCONT     Yes      Yes      Yes      Continued
SIGCHLD    No       No       Yes      Child status changed
SIGTTIN    Yes      Yes      Yes      Stopped (tty input)
---Type <return> to continue, or q <return> to quit---
```

改变信号处理规则

handle signal act	specify GDB actions for <i>signal</i>
print	announce signal
noprint	be silent for signal
stop	halt execution on signal
nostop	do not halt execution
pass	allow your program to handle signal
nopass	do not allow your program to see signal
info signals	show table of signals, GDB action for each

▶ handle SIGPIPE nostop print

```
(gdb) handle SIGPIPE
Signal      Stop      Print    Pass to program Description
SIGPIPE     No        Yes      Yes                Broken pipe
```

多线程调试



info threads

```
(gdb) info threads
  Id  Target Id      Frame
  3  Thread 0xb62b9b40 (LWP 2529) "dconf worker" 0xb76e4424 in __kernel_vsyscall ()
  2  Thread 0xb58ffb40 (LWP 2530) "gdbus" 0xb76e4424 in __kernel_vsyscall ()
* 1  Thread 0xb66a9840 (LWP 2523) "gnome-screensav" 0xb76e4424 in __kernel_vsyscall ()
```

*代表当前线程

切换线程

► thread thread-id

```
(gdb) bt
#0 0xb76e4424 in __kernel_vsyscall ()
#1 0xb6b66460 in poll () from /lib/i386-linux-gnu/libc.so.6
#2 0xb6dd0a3b in g_poll () from /lib/i386-linux-gnu/libglib-2.0.so.0
#3 0xb6dc306e in ?? () from /lib/i386-linux-gnu/libglib-2.0.so.0
#4 0xb6dc352b in g_main_loop_run () from /lib/i386-linux-gnu/libglib-2.0.so.0
#5 0xb72e5ff5 in gtk_main () from /usr/lib/i386-linux-gnu/libgtk-3.so.0
#6 0x0804eed7 in main ()
(gdb) thread 2
[Switching to thread 2 (Thread 0xb58ffb40 (LWP 2530))]
#0 0xb76e4424 in __kernel_vsyscall ()
(gdb) bt
#0 0xb76e4424 in __kernel_vsyscall ()
#1 0xb6b66460 in poll () from /lib/i386-linux-gnu/libc.so.6
#2 0xb6dd0a3b in g_poll () from /lib/i386-linux-gnu/libglib-2.0.so.0
#3 0xb6dc306e in ?? () from /lib/i386-linux-gnu/libglib-2.0.so.0
#4 0xb6dc352b in g_main_loop_run () from /lib/i386-linux-gnu/libglib-2.0.so.0
#5 0xb705f4aa in ?? () from /usr/lib/i386-linux-gnu/libgio-2.0.so.0
#6 0xb6de6673 in ?? () from /lib/i386-linux-gnu/libglib-2.0.so.0
#7 0xb764dd4c in start_thread () from /lib/i386-linux-gnu/libpthread.so.0
#8 0xb6b74bae in clone () from /lib/i386-linux-gnu/libc.so.6
```

对多个线程执行命令

- ▶ `thread apply [thread-id-list | all [-ascending]] command`

```
(gdb) thread apply all bt

Thread 3 (Thread 0xb62b9b40 (LWP 2529)):
#0  0xb76e4424 in __kernel_vsyscall ()
#1  0xb6b66460 in poll () from /lib/i386-linux-gnu/libc.so.6
#2  0xb6dd0a3b in g_poll () from /lib/i386-linux-gnu/libglib-2.0.so.0
#3  0xb6dc306e in ?? () from /lib/i386-linux-gnu/libglib-2.0.so.0
#4  0xb6dc352b in g_main_loop_run () from /lib/i386-linux-gnu/libglib-2.0.so.0
#5  0xb76db134 in ?? ()
   from /usr/lib/i386-linux-gnu/gio/modules/libdconfsettings.so
#6  0xb6de6673 in ?? () from /lib/i386-linux-gnu/libglib-2.0.so.0
#7  0xb764dd4c in start_thread () from /lib/i386-linux-gnu/libpthread.so.0
#8  0xb6b74bae in clone () from /lib/i386-linux-gnu/libc.so.6

Thread 2 (Thread 0xb58ffb40 (LWP 2530)):
#0  0xb76e4424 in __kernel_vsyscall ()
#1  0xb6b66460 in poll () from /lib/i386-linux-gnu/libc.so.6
#2  0xb6dd0a3b in g_poll () from /lib/i386-linux-gnu/libglib-2.0.so.0
#3  0xb6dc306e in ?? () from /lib/i386-linux-gnu/libglib-2.0.so.0
#4  0xb6dc352b in g_main_loop_run () from /lib/i386-linux-gnu/libglib-2.0.so.0
#5  0xb705f4aa in ?? () from /usr/lib/i386-linux-gnu/libgio-2.0.so.0
#6  0xb6de6673 in ?? () from /lib/i386-linux-gnu/libglib-2.0.so.0
#7  0xb764dd4c in start_thread () from /lib/i386-linux-gnu/libpthread.so.0
#8  0xb6b74bae in clone () from /lib/i386-linux-gnu/libc.so.6

Thread 1 (Thread 0xb66a9840 (LWP 2523)):
#0  0xb76e4424 in __kernel_vsyscall ()
```

命名

- ▶ thread name [name]

```
(gdb) info threads
  Id  Target Id      Frame
  3  Thread 0xb62b9b40 (LWP 2529) "dconf worker" 0xb76e4424 in __kernel_vsyscall ()
* 2  Thread 0xb58ffb40 (LWP 2530) "gui" 0xb76e4424 in __kernel_vsyscall ()
  1  Thread 0xb66a9840 (LWP 2523) "gnome-screensav" 0xb76e4424 in __kernel_vsyscall ()
```

命令文件

▶ <https://sourceware.org/gdb/onlinedocs/gdb/Command-Files.html>



Q&A



拍案惊奇——软件调试实战训练营

软件调试高级研习班2017庐山秀峰站

HTTP://001001.ORG/GEDU/ADVDBG2017.PDF

切问而近思
欢迎关注格友公众号

