

Emacs で Hacking

CVS 版 Emacs での GDB + Elscreen

藤原 誠

(株) 絹

Introduction

Emacs で Hacking というと

- c-mode で作成コンパイル

Introduction

Emacs で Hacking というとは

- c-mode で作成コンパイル
- gdb を使ってデバッグ

Introduction

Emacs で Hacking というとは

- c-mode で作成コンパイル
- gdb を使ってデバッグ
- Elscreen を使ってソース探索

gdb というデバッグ系がある

- シェル窓から CUI で使うと熟練が必要

gdb というデバッグ系がある

- シェル窓から CUI で使うと熟練が必要
- `% gdb hello`

gdb というデバッグ系がある

- シェル窓から CUI で使うと熟練が必要
- `% gdb hello`
 - `load/list`

gdb というデバッグ系がある

- シェル窓から CUI で使うと熟練が必要
- `% gdb hello`
 - `load/list`
 - `breakpoint (b) enable/disable/delete`

gdb というデバッグ系がある

- シェル窓から CUI で使うと熟練が必要
- `% gdb hello`
 - `load/list`
 - `breakpoint (b) enable/disable/delete`
 - `run/continue, next/step`

gdb というデバッグ系がある

- シェル窓から CUI で使うと熟練が必要
- `% gdb hello`
 - `load/list`
 - `breakpoint (b) enable/disable/delete`
 - `run/continue, next/step`
 - `finish/until`

gdb というデバッグ系がある

- シェル窓から CUI で使うと熟練が必要
- `% gdb hello`
 - `load/list`
 - `breakpoint (b) enable/disable/delete`
 - `run/continue, next/step`
 - `finish/until`
 - `bt (back trace)`

gdb というデバッグ系がある

- シェル窓から CUI で使うと熟練が必要
- `% gdb hello`
 - `load/list`
 - `breakpoint (b) enable/disable/delete`
 - `run/continue, next/step`
 - `finish/until`
 - `bt (back trace)`
 - `p(print) p/x ptype(struct) x(memory)`

gdb というデバッグ系がある

- シェル窓から CUI で使うと熟練が必要
- `% gdb hello`
 - `load/list`
 - `breakpoint (b) enable/disable/delete`
 - `run/continue, next/step`
 - `finish/until`
 - `bt (back trace)`
 - `p(print) p/x ptype(struct) x(memory)`
 - `info b(breakpoint)/info reg`

`gdb` というデバッグ系がある

- シェル窓から CUI で使うと熟練が必要
- `% gdb hello`
 - `load/list`
 - `breakpoint (b) enable/disable/delete`
 - `run/continue, next/step`
 - `finish/until`
 - `bt (back trace)`
 - `p(print) p/x ptype(struct) x(memory)`
 - `info b(reakpoint)/info reg`
 - `watch`

gdb というデバッグ系がある

- シェル窓から CUI で使うと熟練が必要
- `% gdb hello`
 - `load/list`
 - `breakpoint (b) enable/disable/delete`
 - `run/continue, next/step`
 - `finish/until`
 - `bt (back trace)`
 - `p(print) p/x ptype(struct) x(memory)`
 - `info b(reakpoint)/info reg`
 - `watch`
 - `quit`

今までの Emacs だと

M-x gdb

- Emacs の中から M-x gdb とすると、二画面でソース表示などがあって便利

今までの Emacs だと

M-x gdb

- Emacs の中から M-x gdb とすると、二画面でソース表示などがあって便利
- (二つの窓)

今までの Emacs だと

M-x gdb

- Emacs の中から M-x gdb とすると、二画面でソース表示などがあって便利
- (二つの窓)
- 1. 操作卓 (Console)

今までの Emacs だと

M-x gdb

- Emacs の中から M-x gdb とすると、二画面でソース表示などがあって便利
- (二つの窓)
 - 1. 操作卓 (Console)
 - 2. ソース窓

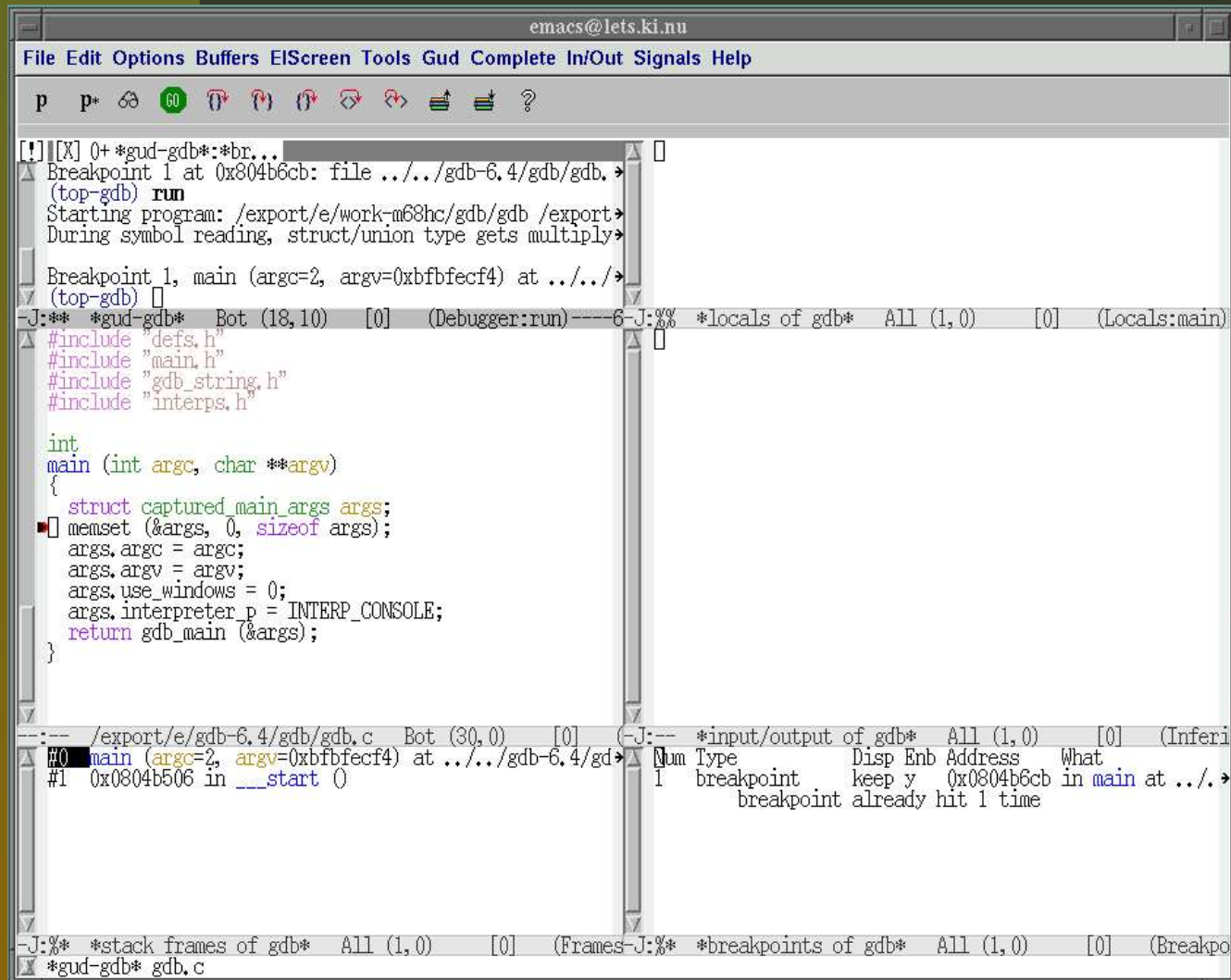
今までの Emacs だと

M-x gdb

- Emacs の中から M-x gdb とすると、二画面でソース表示などがあって便利
- (二つの窓)
 - 1. 操作卓 (Console)
 - 2. ソース窓
- (実はこれでも結構便利、しかし)

CVS 版の Emacs から

M-x gdb すると 6 つの画面が開く



The screenshot shows the Emacs editor interface with several windows open. The main window displays the source code of the `main` function in `gdb.c`. Other windows show GDB status, breakpoints, and stack frames.

```
emacs@lets.ki.nu
File Edit Options Buffers ElScreen Tools Gud Complete In/Out Signals Help

p p* 60

[!] [X] 0+ *gud-gdb*:br...
Breakpoint 1 at 0x804b6cb: file ../../gdb-6.4/gdb/gdb.
(top-gdb) run
Starting program: /export/e/work-m68hc/gdb/gdb /export/
During symbol reading, struct/union type gets multiply
Breakpoint 1, main (argc=2, argv=0xbfbfecf4) at ../../
(top-gdb)
-J:*** *gud-gdb* Bot (18,10) [0] (Debugger:run)-----6-J:%% *locals of gdb* All (1,0) [0] (Locals:main)
#include "defs.h"
#include "main.h"
#include "gdb_string.h"
#include "interps.h"

int
main (int argc, char **argv)
{
  struct captured_main_args args;
  memset (&args, 0, sizeof args);
  args.argc = argc;
  args.argv = argv;
  args.use_windows = 0;
  args.interpreter_p = INTERP_CONSOLE;
  return gdb_main (&args);
}

--- /export/e/gdb-6.4/gdb/gdb.c Bot (30,0) [0] (-J:-- *input/output of gdb* All (1,0) [0] (Inferi
#0 main (argc=2, argv=0xbfbfecf4) at ../../gdb-6.4/gd
#1 0x0804b506 in __start ()
Num Type Disp Enb Address What
1 breakpoint keep y 0x0804b6cb in main at ../../
breakpoint already hit 1 time

-J:*** *stack frames of gdb* All (1,0) [0] (Frames-J:*** *breakpoints of gdb* All (1,0) [0] (Breakpo
*gud-gdb* gdb.c
```

CVS 版の Emacs から

6つの画面を左側から縦に見て行くと、

- 1. 操作卓 (console)

CVS 版の Emacs から

6つの画面を左側から縦に見て行くと、

- 1. 操作卓 (console)
- 2. Source

CVS 版の Emacs から

6 つの画面を左側から縦に見て行くと、

- 1. 操作卓 (console)
- 2. Source
- 3. Stack Frame

CVS 版の Emacs から

6つの画面を左側から縦に見て行くと、

- 1. 操作卓 (console)
- 2. Source
- 3. Stack Frame
- 4. Variable 変数

CVS 版の Emacs から

6つの画面を左側から縦に見て行くと、

- 1. 操作卓 (console)
- 2. Source
- 3. Stack Frame
- 4. Variable 変数
- 5. I/O 目的プログラム入出力

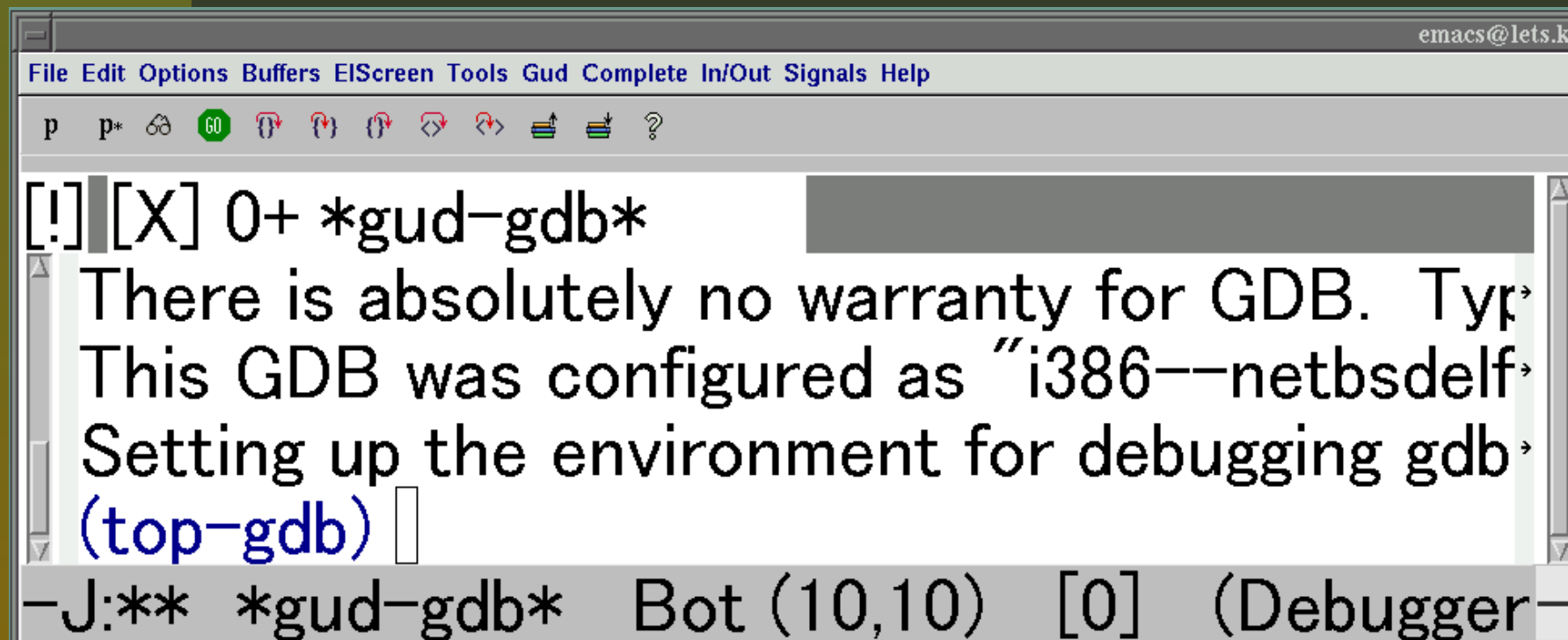
CVS 版の Emacs から

6つの画面を左側から縦に見て行くと、

- 1. 操作卓 (console)
- 2. Source
- 3. Stack Frame
- 4. Variable 変数
- 5. I/O 目的プログラム入出力
- 6. BreakPoint ブレークポイント一覧

1/6 Console

- 操作卓では CUI と同じ操作が出来る

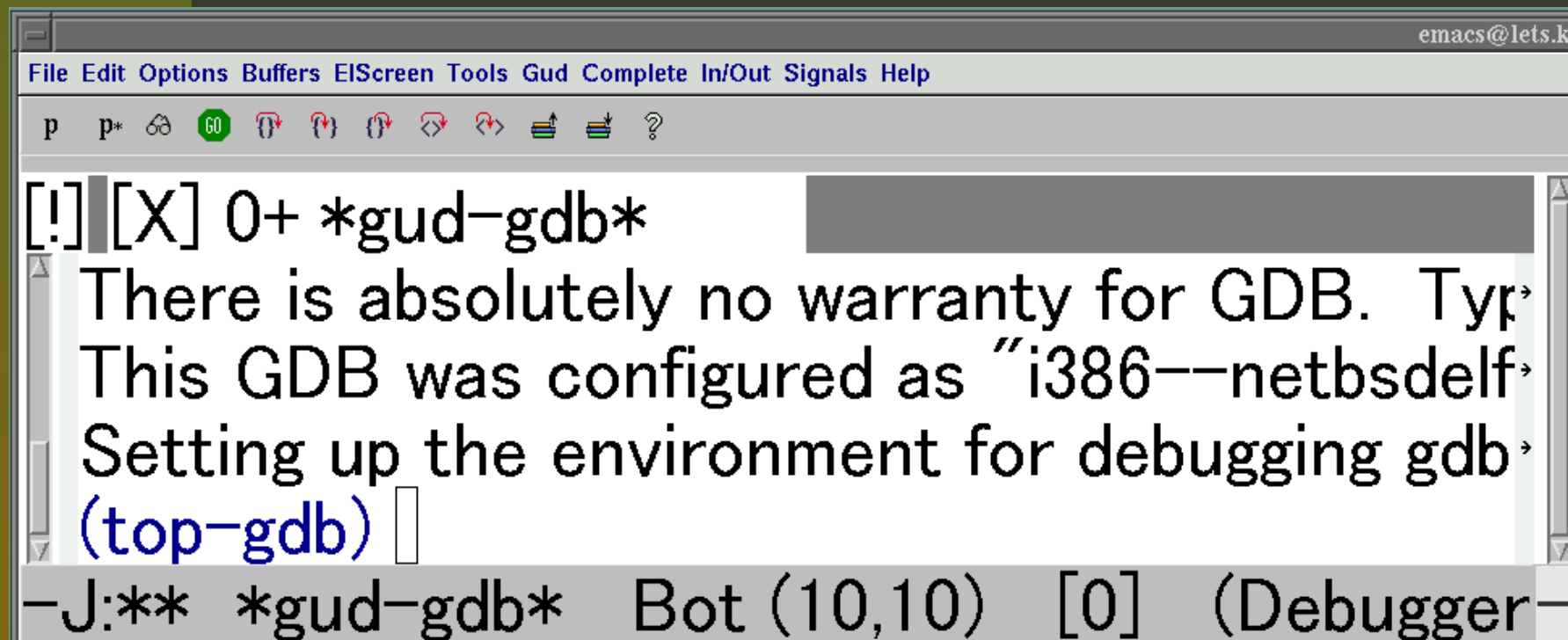


The screenshot shows an Emacs window titled 'emacs@lets.k'. The menu bar includes 'File Edit Options Buffers ElScreen Tools Gud Complete In/Out Signals Help'. The toolbar contains various icons, including a green circle with '60'. The main text area displays the following content:

```
[!] [X] 0+ *gud-gdb*  
There is absolutely no warranty for GDB. Typ  
This GDB was configured as "i386--netbsdelf  
Setting up the environment for debugging gdb  
(top-gdb) |  
-J:** *gud-gdb* Bot (10,10) [0] (Debugger-
```

1/6 Console

- 操作卓では CUI と同じ操作が出来る



The screenshot shows an Emacs window titled 'emacs@lets.k'. The menu bar includes 'File Edit Options Buffers ElScreen Tools Gud Complete In/Out Signals Help'. The toolbar contains various icons, including a green circle with '60'. The main text area displays the following content:

```
[!] [X] 0+ *gud-gdb*  
There is absolutely no warranty for GDB. Typ  
This GDB was configured as "i386--netbsdelf  
Setting up the environment for debugging gdb  
(top-gdb) |  
-J:** *gud-gdb* Bot (10,10) [0] (Debugger-
```

2/6 Source

Source 窓では:

- 「C-x 空白」でブレークポイント (bp) の設定

2/6 Source

Source 窓では:

- 「C-x 空白」でブレークポイント (bp) の設定
- ブレークポイントには赤丸

2/6 Source

Source 窓では:

- 「C-x 空白」でブレークポイント (bp) の設定
- ブレークポイントには赤丸
- bp や step/next で停止した行の表示

2/6 Source

Source 窓では:

- 「C-x 空白」でブレークポイント (bp) の設定
- ブレークポイントには赤丸
- bp や step/next で停止した行の表示
- 次に実行する行には黒三角

2/6 Source

Source 窓では:

- 「C-x 空白」でブレークポイント (bp) の設定
- ブレークポイントには赤丸
- bp や step/next で停止した行の表示
- 次に実行する行には黒三角
- 編集も可

2/6 Source

Source 窓では:

- 「C-x 空白」でブレークポイント (bp) の設定
- ブレークポイントには赤丸
- bp や step/next で停止した行の表示
- 次に実行する行には黒三角
- 編集も可
- Stack Frame を変更すると、それに応じて表示が移る

2. Source (window)

```
int
main (int argc, char **argv)
{
    struct captured_main_args args;
    • memset (&args, 0, sizeof args);
    args.argc = argc;
    args.argv = argv;
    args.use_windows = 0;
    args.interpreter_p = INTERP_CONSOLE;
    return gdb_main (&args);
}
```

---:--- /export/g/src/gdb/gdb.c 80% (30,0) [C-

3/6 StackFrame

スタックフレームには:

- 実行を停止した時の呼出し関係 (Stack Frame) を表示

```
---:--- /export/e/gdb-6.4/sim/m68hc11/interp.c
#0  sim_board_reset (sd=0x82e0000) at ../.../...
#1  0x08138c8b in sim_prepare_for_program (sc
#2  0x08138dcd in sim_open (kind=SIM_OPEN_
#3  0x08067791 in gdbsim_open (args=0x0, fro
#4  0x08068f06 in do_cfunc (c=0x828f780, arg
#5  0x0806a8d2 in cmd_func (cmd=0x828f780,
-J:%* *stack frames of gdb* Top (1,0) [0]
```

3/6 StackFrame

スタックフレームには:

- 実行を停止した時の呼出し関係 (Stack Frame) を表示
- Stack Frame を変更すると、それに応じて Source 側の表示が移る

```
---:--- /export/e/gdb-6.4/sim/m68hc11/interp.c
#0  sim_board_reset (sd=0x82e0000) at ../..../>
#1  0x08138c8b in sim_prepare_for_program (sc>
#2  0x08138dcd in sim_open (kind=SIM_OPEN_>
#3  0x08067791 in gdbsim_open (args=0x0, fro>
#4  0x08068f06 in do_cfunc (c=0x828f780, arg>
#5  0x0806a8d2 in cmd_func (cmd=0x828f780,>
-J:%* *stack frames of gdb* Top (1,0) [0]
```

4/6 変数

- 実行を停止した時のローカル変数名と値を表示

4/6 変数

- 実行を停止した時のローカル変数名と値を表示

5/6 I/O

- デバッグ対象になっているプログラムの入出力窓

```
J:%% *locals of gdb* All (1,0) [0] (Locals:
GNU gdb 6.4
Copyright 2005 Free Software Foundation, Inc
GDB is free software, covered by the GNU Ge
welcome to change it and/or distribute copies
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Typ
This GDB was configured as "--host=i386-un
Setting up the environment for debugging gdb
(
  )
```

5/6 I/O

- デバッグ対象になっているプログラムの入出力窓
- CUI や 21.4 Emacs の版では卓 (Console) と混ってしまう

```
J:%% *locals of gdb* All (1,0) [0] (Locals:
GNU gdb 6.4
Copyright 2005 Free Software Foundation, Inc
GDB is free software, covered by the GNU Ge
welcome to change it and/or distribute copies
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Typ
This GDB was configured as "--host=i386-un
Setting up the environment for debugging gdb
(
  )
```

6/6 BreakPoint

- ブレークポイント一覧を表示

```
Num Type      Disp Enb Address  What
1  breakpoint  keep y  0x0804b6cb in main
   breakpoint already hit 1 time
2  breakpoint  keep y  0x0813853c in sim_k
   breakpoint already hit 1 time

-J:%* *breakpoints of gdb*  All (1,0)  [0] (B
```

6/6 BreakPoint

- ブレークポイント一覧を表示
- enable/disable を変更 (空白キー)

```
Num Type          Disp Enb Address      What
1  breakpoint      keep y  0x0804b6cb in main
   breakpoint already hit 1 time
2  breakpoint      keep y  0x0813853c in sim_k
   breakpoint already hit 1 time

-J:%* *breakpoints of gdb* All (1,0) [0] (B
```

Tool Bar

- 画面上部に ToolBar もある



What is CVS version ?

この環境を使うには:

- Emacs CVS 版 (22.0.50 と表示されるもの) が必要

What is CVS version ?

この環境を使うには:

- Emacs CVS 版 (22.0.50 と表示されるもの) が必要
- CVS 版って何？

What is CVS version ?

この環境を使うには:

- Emacs CVS 版 (22.0.50 と表示されるもの) が必要
- CVS 版って何?
- pkgsrc/editors/emacs に入っているのは 21.4

What is CVS version ?

この環境を使うには:

- Emacs CVS 版 (22.0.50 と表示されるもの) が必要
- CVS 版って何?
- pkgsrc/editors/emacs に入っているのは 21.4
- 開発中の版も公開されていて名前は (今なら)
22.0.50

What is CVS version ?

この環境を使うには:

- Emacs CVS 版 (22.0.50 と表示されるもの) が必要
- CVS 版って何?
- pkgsrc/editors/emacs に入っているのは 21.4
- 開発中の版も公開されていて名前は (今なら) 22.0.50
- そこで cvs から貰って来て make bootstrap

```
cvs -d :pserver:anoncvs@cvs.sv.gnu.org:/sources/emacs co emacs
```

```
mkdir work
```

```
cd work
```

```
../emacs/configure
```

```
make bootstrap
```

```
sudo make install
```

tips – .gdbinit

.gdbinit に手続きを書いておける

- 操作卓で入力するのと同じ文法

tips – .gdbinit

.gdbinit に手続きを書いておける

- 操作卓で入力するのと同じ文法
- 実は Open Source には含まれていることが多い

tips – .gdbinit

.gdbinit に手続きを書いておける

- 操作卓で入力するのと同じ文法
- 実は Open Source には含まれていることが多い
- gdb を実行するディレクトリに置いておく

tips – .gdbinit

.gdbinit に手続きを書いておける

- 操作卓で入力するのと同じ文法
- 実は Open Source には含まれていることが多い
- gdb を実行するディレクトリに置いておく
- src が分散している時には `dir ../src/hoge/`

tips – .gdbinit

.gdbinit に手続きを書いておける

- 操作卓で入力するのと同じ文法
- 実は Open Source には含まれていることが多い
- gdb を実行するディレクトリに置いておく
- src が分散している時には `dir ../src/hoge/`
- 記述例は

```
set args ../input-file.s (引数の指定)
b main (main に breakpoint)
run (実行開始)
```

tips – .gdbinit

.gdbinit に手続きを書いておける

- 操作卓で入力するのと同じ文法
- 実は Open Source には含まれていることが多い
- gdb を実行するディレクトリに置いておく
- src が分散している時には `dir ../src/hoge/`
- 記述例は

```
set args ../input-file.s (引数の指定)
b main (main に breakpoint)
run (実行開始)
```


tips – directory

- M-x gdb で directory/program 等と指定すると directory に移動してしまう

tips – directory

- M-x gdb で directory/program 等と指定すると directory に移動してしまう
- つまり (set args で) 引数を指定する時の PATH にも注意

tips – directory

- M-x gdb で `directory/program` 等と指定すると `directory` に移動してしまう
- つまり (set args で) 引数を指定する時の PATH にも注意
- 例えば `set args ../input-filename`

tips – directory

- M-x gdb で `directory/program` 等と指定すると `directory` に移動してしまう
- つまり (`set args` で) 引数を指定する時の `PATH` にも注意
- 例えば `set args ../input-filename`
- そこで Emacs 起動前から `directory` に移動しておく と 分りやすい

tips – directory

- M-x gdb で `directory/program` 等と指定すると `directory` に移動してしまう
- つまり (`set args` で) 引数を指定する時の `PATH` にも注意
- 例えば `set args ../input-filename`
- そこで Emacs 起動前から `directory` に移動しておく と 分りやすい

Elscreen-GF

Elscreen-GF

- id-utils を Emacs から使えるようにしたもの

Elscreen-GF

Elscreen-GF

- id-utils を Emacs から使えるようにしたもの
- <http://www.morishima.net/naoto/software/elscreen/>

Elscreen-GF

Elscreen-GF

- id-utils を Emacs から使えるようにしたもの
- <http://www.morishima.net/naoto/software/elscreen/>
- pkgsrc/devel/id-utils を入れておく

Elscreen-GF

Elscreen-GF

- id-utils を Emacs から使えるようにしたもの
- <http://www.morishima.net/naoto/software/elscreen/>
- pkgsrc/devel/id-utils を入れておく
- 場合によっては mkid で ID という名前の tag を作っておく

Elscreen-GF

Elscreen-GF

- id-utils を Emacs から使えるようにしたもの
- <http://www.morishima.net/naoto/software/elscreen/>
- pkgsrc/devel/id-utils を入れておく
- 場合によっては mkid で ID という名前の tag を作っておく
- 探したい関数・変数・マクロ名等にカーサを移動

Last slide

- この発表資料は prosper を使って pLaTeX で作成し xpdf または Acrobat Reader で表示しています。
<http://mechanics.civil.tohoku.ac.jp/soft/node10.html>
<http://prosper.sourceforge.net/>

Last slide

- この発表資料は prosper を使って pLaTeX で作成し xpdf または Acrobat Reader で表示しています。
<http://mechanics.civil.tohoku.ac.jp/soft/node10.html>
<http://prosper.sourceforge.net/>
- ご静聴ありがとうございました。