

# Emacs で Hacking

## *CVS 版 Emacs で の GDB + Elscreen*

藤原 誠

(株) 絹

<http://www.ki.nu/~makoto/e/emacs-gdb.pdf>

# Introduction

---

Emacs で Hacking というとは

- c-mode で作成・編集とコンパイル

# Introduction

---

Emacs で Hacking というとは

- c-mode で作成・編集とコンパイル
- M-x gdb を使ってデバッグ

# Introduction

---

Emacs で Hacking というとは

- c-mode で作成・編集とコンパイル
- M-x 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**

# 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 `gdb.c` with a breakpoint set at line 30. Other windows show the GDB console output, including the start of the program and the hit of the breakpoint. The Emacs title bar indicates the user is `emacs@lets.ki.nu`.

```
emacs@lets.ki.nu
File Edit Options Buffers ElScreen Tools Gud Complete In/Out Signals Help
p p* 60 [GDB icons] ?
[!] [[X] 0+ *gud-gdb*: *br...
Breakpoint 1 at 0x804b6cb: file ../../gdb-6.4/gdb/gdb.c
(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 ../../gdb-6.4/gdb/gdb.c:30:
(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] (Inferior)
#0 main (argc=2, argv=0xbfbfecf4) at ../../gdb-6.4/gdb/gdb.c:30:
#1 0x0804b506 in __start ()
Num Type Disp Enb Address What
1 breakpoint keep y 0x0804b6cb in main at ../../gdb-6.4/gdb/gdb.c:30:
breakpoint already hit 1 time

-J:* *stack frames of gdb* All (1,0) [0] (Frames:J:-- *breakpoints of gdb* All (1,0) [0] (Breakpoints)
```

# 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

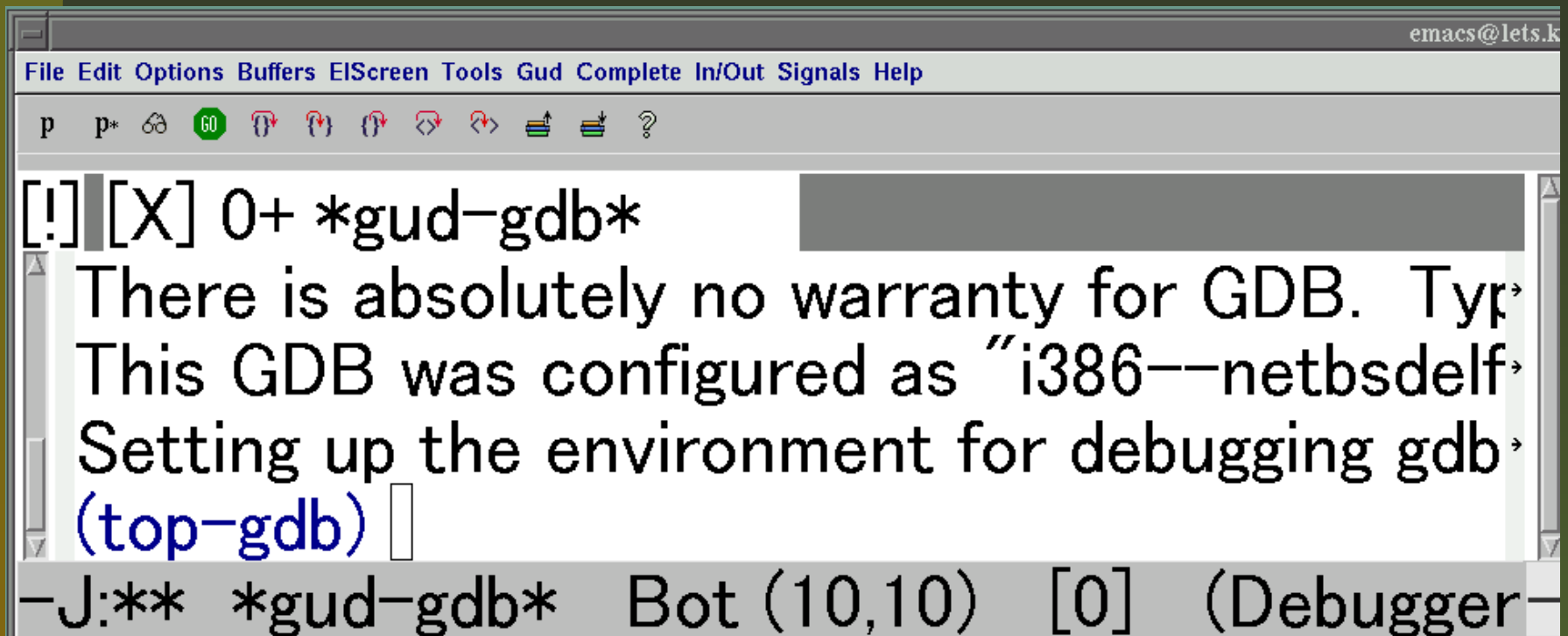
# 1/6 Console

---

- 操作卓では CUI と同じ操作が出来る

# 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 icons for "p", "p\*", "60", and several other symbols. The main text area displays the GDB console output:

```
[!] [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 で停止した行を表示

## 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 で停止した行を表示
- 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 で停止した行を表示
- 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 で停止した行を表示
- 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



# 5/6 5 番目は 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
(top-gdb) 
```

# 5/6 5 番目は 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) と混る

```
-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
(top-gdb) 
```

# 6/6 最後の BreakPoint 窓では

- 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 最後の BreakPoint 窓では

- 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
```

## 6/6 最後の BreakPoint 窓では

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

```
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)
```



# make する時には

---

対象プログラムを make する時には

- `-g` または `-gdwarf-2` を付けておく

# make する時には

---

対象プログラムを make する時には

- `-g` または `-gdwarf-2` を付けておく
- 上の項目は (必須) ただしいは付いている

# make する時には

---

対象プログラムを make する時には

- `-g` または `-gdwarf-2` を付けておく
- 上の項目は(必須) ただしいは付いている
- `-O` 等は付けない

# make する時には

---

対象プログラムを make する時には

- `-g` または `-gdwarf-2` を付けておく
- 上の項目は (必須) ただしいていは付いている
- `-O` 等は付けない
- 上の項目は付けたままでも一応何とかなる。外すのは ..

# 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 – 画面がおかしくなったら

---

- 一番上のメニューの Gud

# tips – 画面がおかしくなったら

---

- 一番上のメニューの Gud
- ...GDB-UI

# tips – 画面がおかしくなったら

---

- 一番上のメニューの Gud
- ...GDB-UI
- .....Restore Window Layout

# Elscreen-GF って何

---

(話は違って) Elscreen-GF

- id-utils という CUI 版の 識別子検索がある

# Elscreen-GFって何

---

(話は違って) Elscreen-GF

- id-utils という CUI 版の 識別子検索がある
- C/C++ の関数名、変数名、マクロ名、(構造体) 要素名でソース内を検索する

# Elscreen-GFって何

---

(話は違って) Elscreen-GF

- id-utils という CUI 版の 識別子検索がある
- C/C++ の関数名、変数名、マクロ名、(構造体) 要素名でソース内を検索する
  - mkid 索引を作る

# Elscreen-GFって何

---

(話は違って) Elscreen-GF

- id-utils という CUI 版の 識別子検索がある
- C/C++ の関数名、変数名、マクロ名、(構造体) 要素名でソース内を検索する
  - mkid 索引を作る
  - lid 譜名を表示

# Elscreen-GFって何

(話は変って) Elscreen-GF

- id-utils という CUI 版の 識別子検索がある
- C/C++ の関数名、変数名、マクロ名、(構造体) 要素名でソース内を検索する
  - mkid 索引を作る
  - lid 譜名を表示
  - gid ソース内を grep



# Elscreen-GFって何

(話は変って) Elscreen-GF

- id-utils という CUI 版の 識別子検索がある
- C/C++ の関数名、変数名、マクロ名、(構造体) 要素名でソース内を検索する
  - mkid 索引を作る
  - lid 譜名を表示
  - gid ソース内を grep
- その id-utils を Emacs から使えるようにしたもの

# Elscreen-GF(準備)

---

- <http://www.morishima.net/~naoto/software/elscreen/>から elscreen-GF をもらって来て、lisp directory に置く  
(場合によってはバイトコンパイル)

# Elscreen-GF(準備)

---

- <http://www.morishima.net/~naoto/software/elscreen/>から elscreen-GF をもらって来て、lisp directory に置く  
(場合によってはバイトコンパイル)
- pkgsrc/devel/id-utils/ を入れておく

# Elscreen-GF(準備)

- <http://www.morishima.net/~naoto/software/elscreen/> から elscreen-GF をもらって来て、lisp directory に置く  
(場合によってはバイトコンパイル)
- pkgsrc/devel/id-utils/ を入れておく
- 場合によってはソースの一番上で `mkid` と入力し ID という名前の tag を作っておく

```
cd src
```

```
mkid
```

# Elscreen-GF(使い方)

---

## Elscreen-GF

- 探したい関数・変数・マクロ名等にカーサを移動

# Elscreen-GF(使い方)

---

## Elscreen-GF

- 探したい関数・変数・マクロ名等にカーサを移動
- `elscreen-gf-idutils-gid` に割当ててあるキーを入力  
初期値は C-z C-g G

僕の場合は次のように設定している

```
(global-set-key [insert] 'elscreen-gf-idutils-gid)
```

# Elscreen-GF(使い方)

## Elscreen-GF

- 探したい関数・変数・マクロ名等にカーサを移動
- `elscreen-gf-idutils-gid` に割当ててあるキーを入力  
初期値は `C-z C-g G`  
僕の場合は次のように設定している  
(`global-set-key [insert] 'elscreen-gf-idutils-gid`)
- 表示された一覧から選んで `o`

# まとめ

---

- CVS 版の Emacs で gdb を使うと 6 画面で楽しい



# まとめ

---

- CVS 版の Emacs で gdb を使うと 6 画面で楽しい
- その CVS 版は `cvs co` して `make bootstrap` する

# まとめ

---

- CVS 版の Emacs で gdb を使うと 6 画面で楽しい
- その CVS 版は cvs co して make bootstrap する
- Elscreen-GF を使うと C/C++ の識別子検索の効率が良い

# 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。