Mucking out and cleaning up

Professionals deal with a lot of Muck!

Topics Covered

- Why software rots
- Tools for cleanup
- The cleanup process
- Paranoia

  or "Just because your paranoid, it doesn't mean they aren't out to get you.
- How to keep things clean

Why Software Rots

Priorities

Commercial Developers

1) Code
2) Debug
3) Add Comments
4) Think about the project and design it.

Management

Give developers a new assignment before they finish ~~#2~~.

#1

Correcting the Problem

- Define the problem and *write it down.*

- Think then code

The order is not optional

- Communicating with Management

"Why is there never enough time to do things right, but always enough time to do things over."

Good Development Cycle

- Write a good specification which can become
- A good program design which can become
- A good set of comments which are a framework for:
- A good program

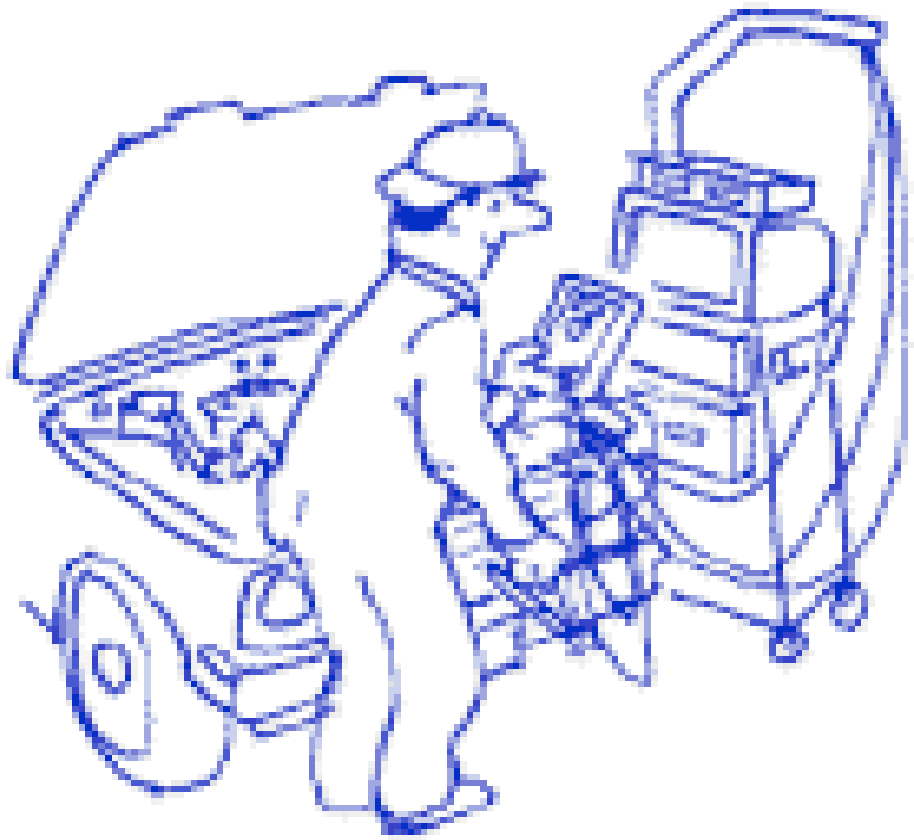
Standards. We don't need no stinking standards!

- Myth: Our programmers don't need a style sheet, they are all experts and know how to right a good program.
-
- Fact: There's more than one programming style. Without a style sheet, all will be used resulting in confusion.

No Standard

```
static void display_mem(){
int mem_size = 24 * K;
    int bytes_left = mem_size -€mem_used;

   if (external_flag)
       {
       init_external();
       mem_ptr = external_flash;
       }
       if (internal) {
           pack_internal();
       }
    if (free_flag) {
      if (bytes) {
      printf( "%ld\n", dir_header.bytes_remaining);
} else {
  printf( "%ld\n", dir_header.block_remaining)
    }
}
printf( "   entry count: %hu\n",
dir_header.entry_count );
}
```

Standard Units



- Emissions are measured in:
- Grams / Mile
-

Unit Problems

Lack of standards can lead to hacks

Not Standards Units Results In:

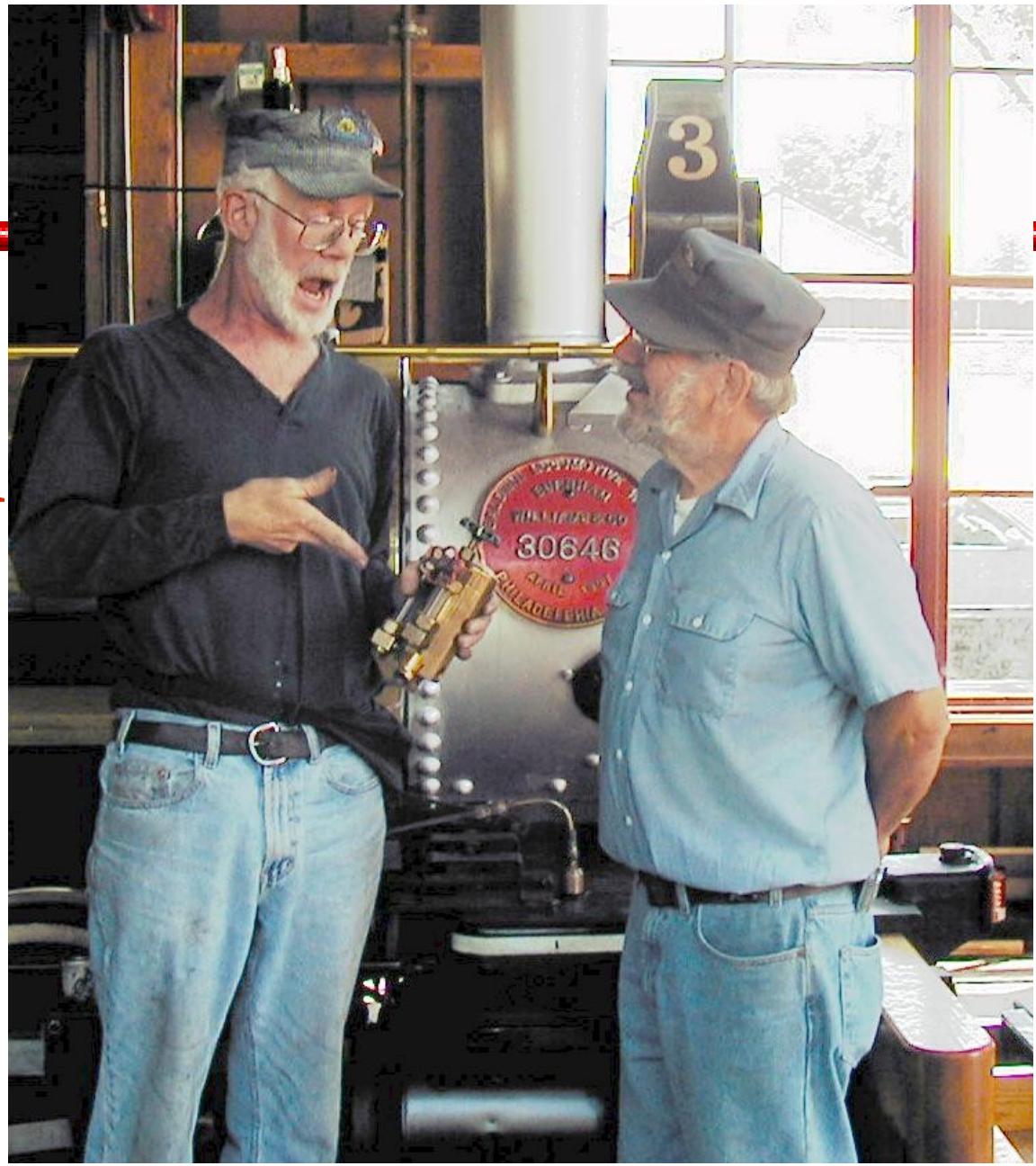
```
/*-----------------------------*
 * I have no idea what the input *
  * units are nor do I have any  *
 * idea what the output units    *
 * but I do know if I divide by  *
 * three the plots look about    *
 * the right size.               *
  *-----------------------------*
                              */
```

What design document?

- Myth: Everyone knows what we're trying to do. A design is not needed.
  -
  -

Uncommented Code

"Manzel doesn't have records that go back that far."



I don't have time to put in the comments.

- You should know what you're doing before you do it — write it down.
- I don't have reverse engineer your uncommented code.

I'll put in the comments later.

- Later never comes
- People forget what they did. (Sometimes five minutes after they did it.)
- The result of "later" is incomplete and inaccurate comments.

My code is self documenting, I don't need comments.

```
if (this.believe == true) {
  you.hacking = Excessive;
}
```

(If you believe this you've been hacking too much)

The specification is in my head

- Am I supposed to open your head and read the comments?

Our code quality is no worse than the next guy.

The next guy is:

Need I say more?

Commenting: My Biggest Challenge

- A programmer:
- I couldn't discipline
- I couldn't fire.
- and who had absolutely no respect for me as an industry leader.

My Biggest Challenge: My Wife!

- Her: The teacher doesn't make me put in comments.
- Me: I wrote "C Elements of Style" and no wife of mine will turn in uncommented code.

The Legacy Platform Trap

```
#ifdef LOCATE_CALLS
char *save_block(char *block, int size, char *file, int line)
#endif
#ifndef LOCATE_CALLS
char *save_block(char *block, int size)
#endif
{
#ifndef NATIVE_MALLOC
    char *result;
#endif
#ifdef LOG_CALLS
    log_message(file, line, "save_block(%p, %d)", block,
size);
#endif LOG_CALLS
#ifdef NATIVE_MALLOC
    return (memcpy(malloc(size), block, size));
#endif
#ifdef SINGLE_BLOCK_ALLOC
    result = find_memory(size);
    result += sizeof(struct mem_header);
    memcpy(result, block, size);
    return (result);
#endif
#ifdef OOO_INTERFACE
#ifdef LOCATE_CALLS
    return ((*alloc)(file, line, block, size));
#else
```

#ifdef HELL

sys_funct.c -- module (18000) lines

2,000 used for the os simulator

2,000 used for the GUI

2,000 used for flash simulator

1,500 common

==========

7,500 total

Do you really need to support obsolete machines?

Language Differences

```
// Die Hauptroutine. Durch wiederholten Aufrufs dieser
// Routine wird
// die Quelle geparst. Returnwert FALSE bei
// Ende/Fehlern.
BOOL SbiParser::Parse(){
    if( bAbort ) return FALSE;
    EnableErrors();
    Peek();
    // Dateiende?
    if( IsEof() ) {
        // AB #33133: Falls keine
        // Sub angelegt wurde, muss hier
        // der globale Chain abgeschlossen
        // werden!
        // AB #40689: Durch die neue
        // static-Behandlung kann noch
        // ein nGblChain vorhanden sein,
        // daher vorher abfragen
```

Language Differences

- Translation problems can cause difficulties

"Is called by" -> "Is shouted at"

Tower Bridge (Built 1886)

"We didn't restore it, we preserved it."

Preventing Rot

- Think and code *In That Order!*
  - Produce Standards
    - Enforce Them

Why Open Source Rots Less

- Pride of workmanship
- No Schedule Pressure
- Lots of Peer Pressure


Tools
Jim Hocket
Ryan Hocket
Chuck Cross

Tools

- grep
- find/grep
- glimpse
- indent
- Vim
- lxr
- ox (object xref)
- cpp
- debugger
- source navigator

Simple grep example

```
grep regdump *.[ch]
embed.h:#define regdump        Perl_regdump
embed.h:#define regdump(a)
   Perl_regdump(aTHX_ a)
proto.h:PERL_CALLCONV void
   Perl_regdump(pTHX_ regexp* r);
regcomp.c:#  define Perl_regdump my_regdump
regcomp.c:    DEBUG_r(regdump(r));
regcomp.c: - regdump - dump a regexp onto
   Perl_debug_log in vaguely comprehensible
   form
regcomp.c:Perl_regdump(pTHX_ regexp *r)
regexec.c:#  define Perl_regdump my_regdump
```

grep all files command

```
grep regdump * | cat -v | cut -c 1-80
```

`cat -v` **Turns unprintable characters into something readable.**

`cut -c 1-16`

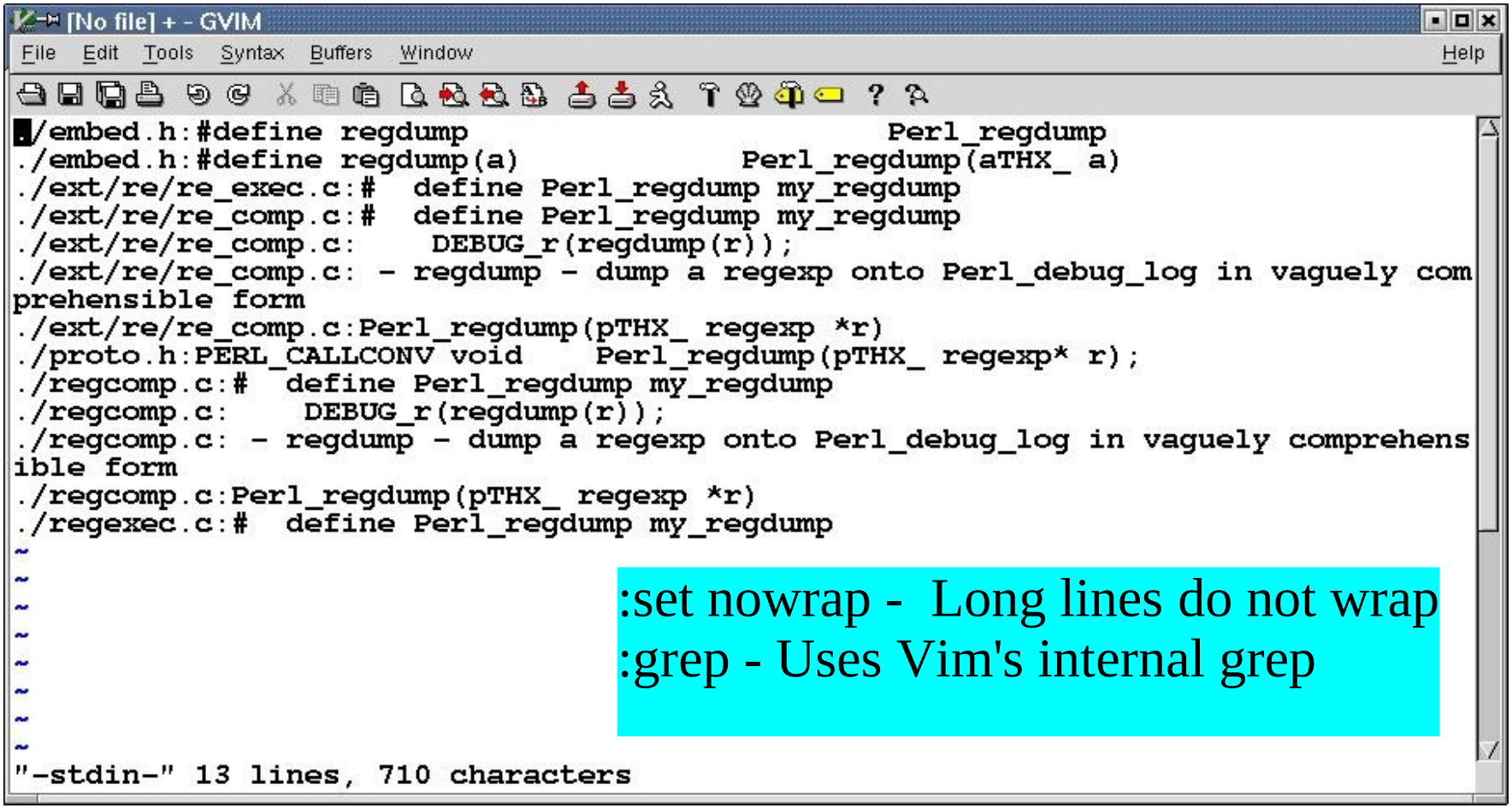
Binary files have long "lines". This command trims them to 80 characters long for viewing and printing.

grep all files example

```
grep regdump * | cat -v | cut -c 1-80
grep: Cross: Is a directory
grep: NetWare: Is a directory
...
embed.fnc:Ap |void•  |regdump  |regexp* r
embed.h:#define regdump       Perl_regdump
embed.h:#define regdump(a)
   Perl_regdump(aTHX_ a)
global.sym:Perl_regdump
libperl.a:^@^PM-^K@^PMPM-^KM-^J^@^PM-^KM-^J^@
libperl.a:^^@^@^@^H^@^@^@P^@^@^@^A^@^@^@^@^@^
miniperl:^Hn^@-^N^^@^LM-3^D^H7^@^@^@^R^@^@^@s
perl:^@R^@^M^@`M-F  ^H1^@^@^@^R^@^M^@M-^WM-^
....
```

grep files and view with Vim

```
grep regdump *.[ch] | cat -v | gvim -
```



find and grep

```
find .
\( -name "*.cpp" -o -name "*.h" \)
-exec fgrep what {} /dev/null \;
```

find . Find starting at current directory

\(\) Group operation

-name "*.cpp" -o -name "*.h"

All C++ *or* H files

-exec Command to execute

find and grep

```
find .
\( -name "*.cpp" -o -name "*.h" \)
-exec fgrep what {} /dev/null \;
```

-exec fgrep Execute fgrep command

{} on the current file

/dev/null Also search /dev/null

\; End of command

fgrep prints the file name only when two files searched, thus */dev/null*

grep and find example

```
find . -name *.[ch] -exec fgrep regdump {}
   /dev/null ;
./embed.h:#define regdump     Perl_regdump
./embed.h:#define regdump(a)Perl_regdump()
./ext/re/re_exec.c:#define Perl_regdump
   my_regdump          •
./ext/re/re_comp.c:#define Perl_regdump
   my_regdump          •
./ext/re/re_comp.c:    DEBUG_r(regdump(r));
./ext/re/re_comp.c: - regdump - dump a regexp
   onto Perl_debug_log in vaguely comprehensible
   form
./ext/re/re_comp.c:Perl_regdump(pTHX_ regexp *r)
....
```

grep -r

```
fgrep -r regdump .
./ext/re/re_comp.c:#  define Perl_regdump
my_regdump
./ext/re/re_comp.c:    DEBUG_r(regdump(r));
./ext/re/re_comp.c: - regdump - dump a regexp
onto Perl_debug_log in vaguely comprehensible
form
./ext/re/re_comp.c:Perl_regdump(pTHX_ regexp *r)
Binary file ./ext/re/re_comp.o matches
./ext/re/re_exec.c:#  define Perl_regdump
my_regdump
Binary file ./lib/auto/re/re.so matches
Binary file ./.swp matches
./embed.fnc:Ap |void   |regdump  |regexp* r
./embed.h:#define regdump      Perl_regdump
```

find + grep vs. grep -r

- **grep -r** is much faster
- **find** gives you better control over the files searched.
- **grep -r** is a GNU feature that's not available in most UNIX **grep** programs.

Glimpse

- Super fast grep.

Advantages

- Extremely fast searches
- Superior text searches

Disadvantages

- Restricted license
- Building of index database takes time
- Reindex needed when program changes

Generating a Glimpse Index

```
find oo_1.0.1_src \
  ( -name *.h -o -name *.c* ) -print |\
   glimpseindex -H /home/sdo/muck/tools -F
```

-H -- Specify directory for the database files

-F -- Read list of files to index from standard in.

Glimpse Search

Perform Search

```
glimpse -H /home/sdo/muck/tools -n linux
```

-H -- Specify the location of the database

-n -- Print the line number where the match occurs

Glimpse Example Index

```
find oo_1.0.1_src \
  ( -name *.h -o -name *.c* ) -print |\
   glimpseindex -H /home/sdo/muck/tools -F

This is glimpseindex version 4.16.2, 2002.

Indexing "oo_1.0.1_src/common/english_us/custom.css" ...
Indexing "oo_1.0.1_src/sbasic/english_us/sbasic.cfg" ...
Indexing "oo_1.0.1_src/parser_i/tokens/tkpcont2.cxx" ...
...
Indexing "oo_1.0.1_src/parser_i/tokens/tkpstam2.cxx" ...
Indexing "oo_1.0.1_src/autodoc/source/tools/tkpchars.cxx"
...
Size of files being indexed = 137736911 B, Total #of files =
7600
```

Glimpse Commands

```
glimpse -H /home/sdo/muck/tools
        -n linux

Your query may search about 24% of the total space!
Continue? (y/n)
oo_1.0.1_src/dbaccess/source/ui/browser/dsbrowserDnD.cxx:
    1279:  *  #65293# linux ambiguity
oo_1.0.1_src/dbaccess/source/ui/dlg/dsselect.cxx: 223:  *
    #65293# cant compile for linux
oo_1.0.1_src/dbaccess/source/ui/dlg/indexdialog.cxx: 913:  *
    Some error checked for linux
oo_1.0.1_src/dbaccess/source/ui/dlg/odbcconfig.cxx: 352:  *
    #65293# cant compile for linux
oo_1.0.1_src/dbaccess/source/ui/dlg/sqlmessage.cxx: 603:  *
    Syntax error with linux compiler
oo_1.0.1_src/svx/source/fmcomp/fmgridif.cxx: 1642:    // the
    same props as in addColumnListeners ... linux has
    problems with global static UStrings, so
```

Indent

- Makes code readable
- Very useful when several styles of coding used
- May not work correctly on really bad code

Indent

```
BOOL SbiParser::Parse(){if(bAbort)return
FALSE;EnableErrors();Peek();if(IsEof()){if(
bNewGblDefs&&nGblChain==0)nGblChain=aGen.Gen(
_JUMP, 0 );return FALSE;}if(IsEoln(eCurTok)){
Next();return TRUE;}if(!bSingleLineIf&&
MayBeLabel(TRUE)){if(!pProc)Error(
SbERR_NOT_IN_MAIN,aSym);else pProc->
GetLabels().Define( aSym );Next();Peek();if(
IsEoln(eCurTok)){Next();return TRUE;}}if(
eCurTok==eEndTok){Next();if(eCurTok!=NIL)
aGen.Statement();return FALSE;}if(eCurTok==
REM){Next();return TRUE;}if(eCurTok==SYMBOL
||eCurTok==DOT){if(!pProc)Error(
SbERR_EXPECTED,SUB ); else{Next();Push(
eCurTok);aGen.Statement();Symbol();}}
```

Indent

```
BOOL SbiParser::Parse ()
{
    if (bAbort)
        return FALSE;
    EnableErrors ();
    Peek ();
    if (IsEof ())
      {
          if (bNewGblDefs && nGblChain == 0)
              nGblChain = aGen.Gen(_JUMP, 0);
          return FALSE;
```

Jeff Code

One of the people I worked with really did code with:

- No comments
- No indents
- No line breaks (unless forced)

He believed that the best way to program was to put as much code on the screen at one time as possible.

Source Navigator

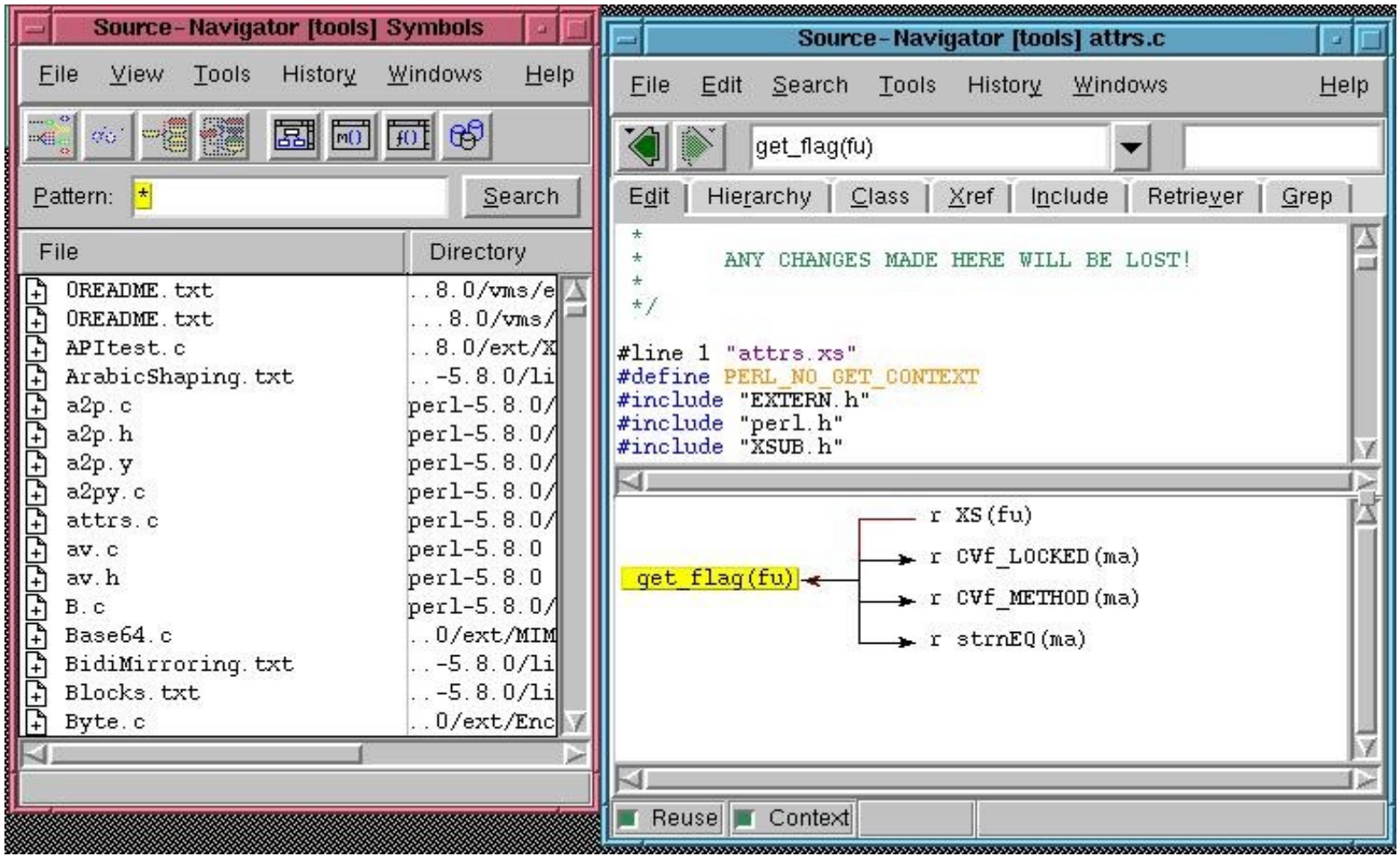
Provides

- Source browsers
- Cross Reference Tool

Limitations

- Building the index is slow
- Difficult to understand command interface

Source Navigator



Vim

Vim is charity-ware.

Give to the Ugandan Children

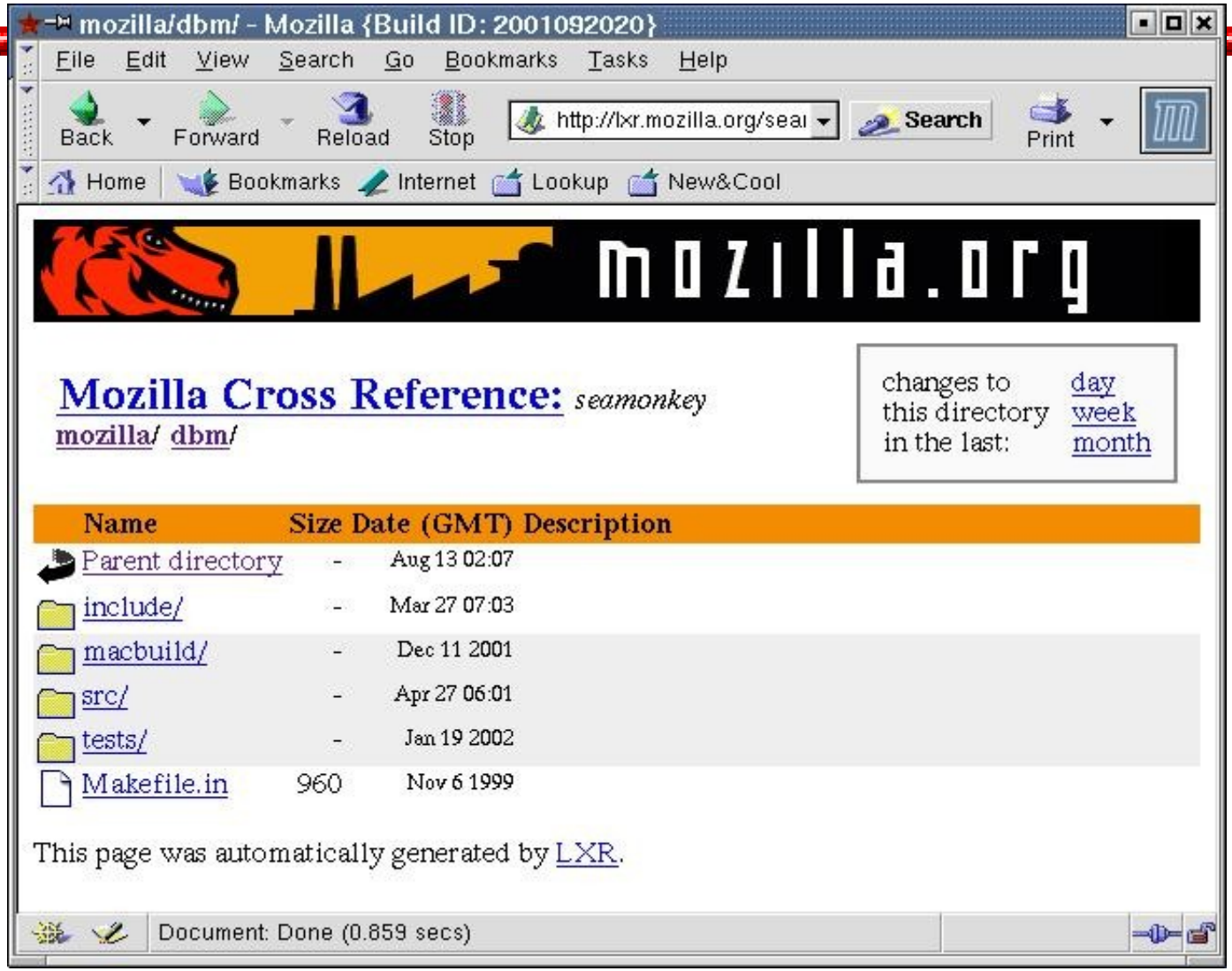
Vim

- :grep
- = (=% is very good)
- *ctags*
- Tag navigation (^], ^T)
- :grep
- Find definition --[d,]d, [D,]D
- Fast searches #, *

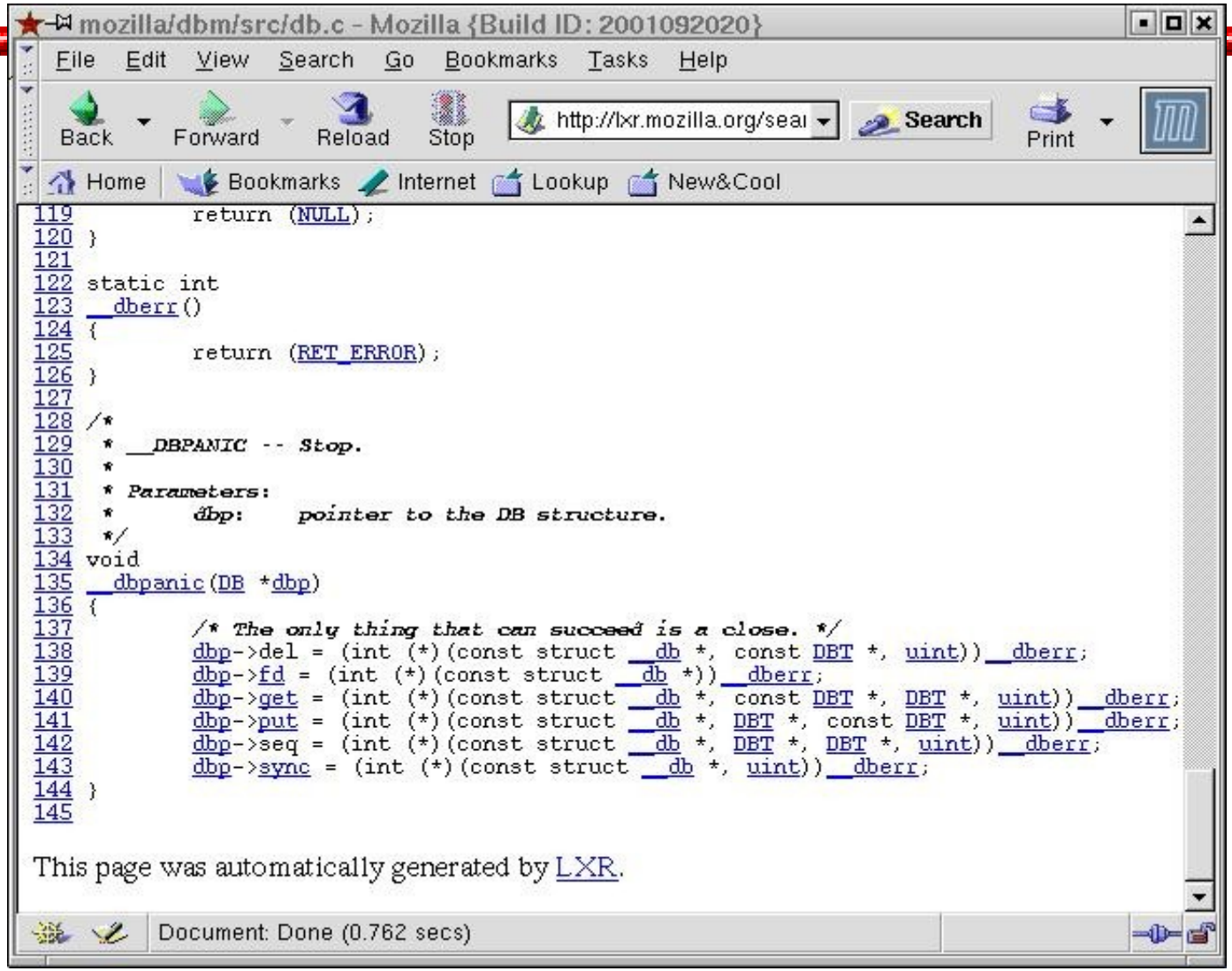
Linux Cross Reference (lxr)

- Available: http://www.lxr.org
- Cross References the Linux Kernel and other source
- But!
  - Requires web server
  - Don't install unless you've know Perl

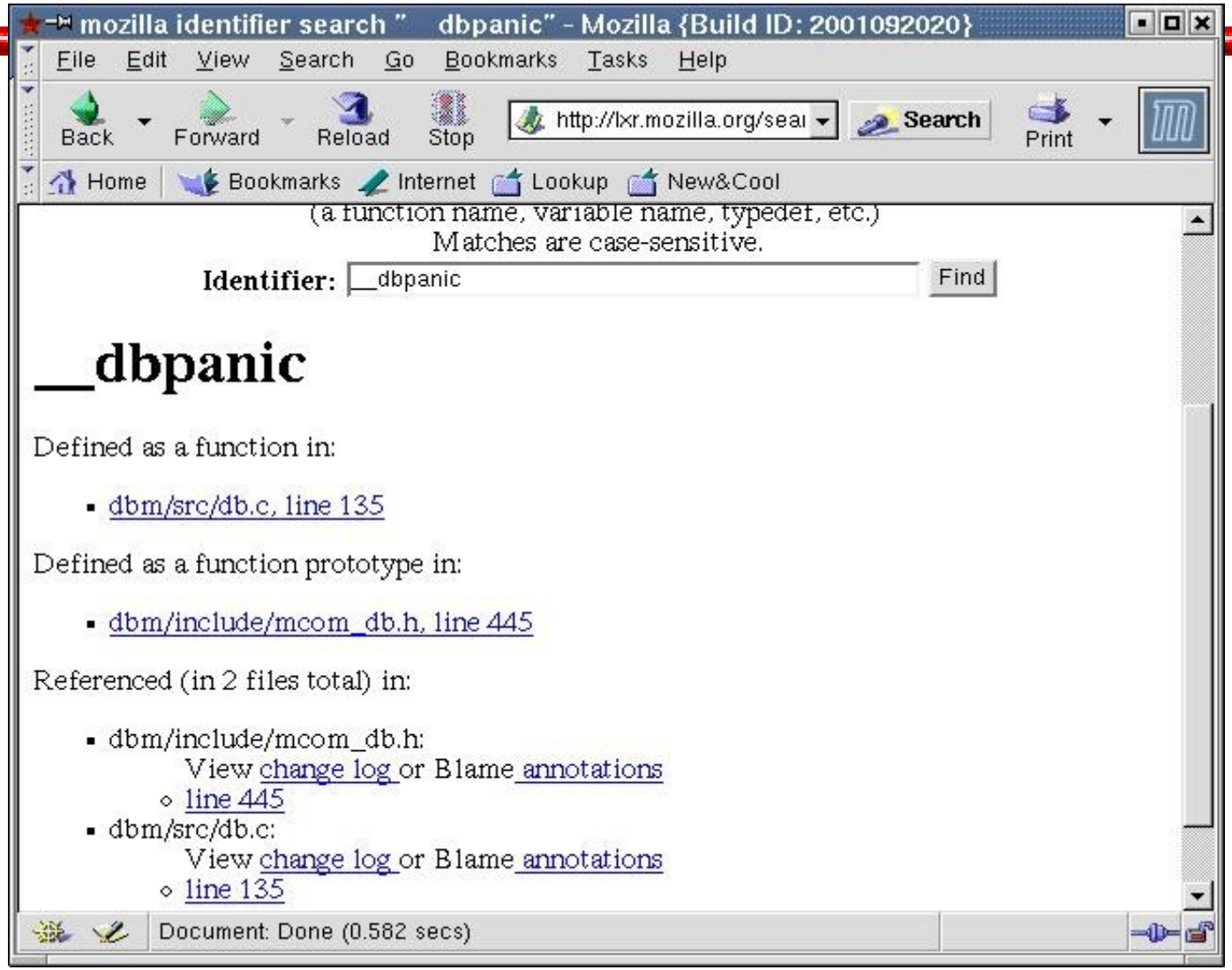
lxr File List



lxr Source View



lxr Identifier Cross Reference



ox (Object Cross Reference)

Generates a cross reference of the object files.

- ox_gen.pl -- Generates data file
- ox.pl -- Finds symbols

Very good for finding out where an undefined symbol is used.

ox (Object Cross Reference)

`perl ox-gen.pl -- Generate index`

Searching for "search_index"

```
$ perl ox.pl search_index
search_index
    Defined: ./get_index.o
    Used: ./main.o ./main_server.o
```

The Pre-processor

- **Takes "#ifdef HELL" code and tells you what's really compiled.**
- **To use:**
  1) **Compile normally, record the data in a log file.**
  2) **Find the compilation command in the log file. Write it to a shell script.**
  3) **Edit the script and change -c (and other flags) to -E (pre-processor output).**

The Pre-processor

```
#ifdef LOCATE_CALLS
char *save_block(char *block, int size, char
*file, int line)
#endif
#ifndef LOCATE_CALLS
char *save_block(char *block, int size)
#endif
{
#ifndef NATIVE_MALLOC
    char *result;
#endif
#ifdef LOG_CALLS
    log_message(file, line, "save_block(%p,
%d)", block, size);
#endif LOG_CALLS
#ifdef NATIVE_MALLOC
    return (memcpy(malloc(size), block, size));
#endif
#ifdef SINGLE_BLOCK_ALLOC
    result = find_memory(size);
    result += sizeof(struct mem_header);
    memcpy(result, block, size);
    return (result);
#endif
#ifdef OOO_INTERFACE
#ifdef LOCATE_CALLS
    return ((*alloc)(file, line, block, size));
#else
    return ((*alloc)(block, size));
#endif
#endif
```

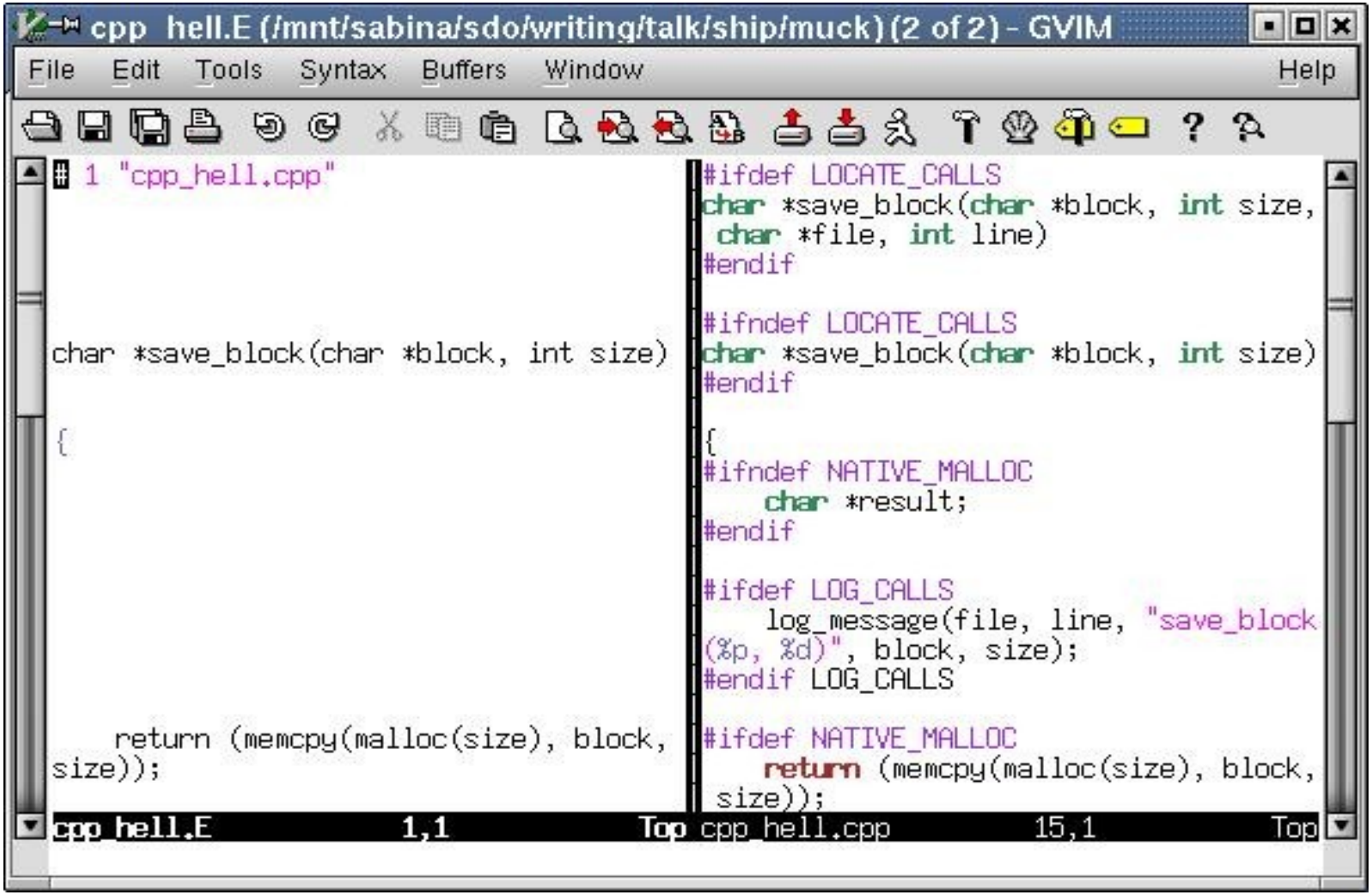
```
# 1 "cpp_hell.cpp"


char *save_block(char *block, int size)

{

    return (memcpy(malloc(size), block, size));

}
```

Pre-processor and Vim

1) Edit files with *gvim source pre-file*
2) Set the scrollbind option (**:set scrollbind**)
3) Split windows vertically (**:vsplit**) and go to the next file (**:next**)

You now have two windows side by side which the source file and the pre-processed file.

Vim and the pre-processor



Dynamic Analysis

- Debugger
- Log Files and Instrumentation
- Unit Tests
- Playing in the Sandbox

Debugger

- Can see what is really going on
- Lets you watch the internals of the program as it does it's job.
- Requires test environment
- Sometimes it's hard to get the program to execute the code

Logs

Log Code

```
void log_msg(char *fmt, ...)

{
    char full_string[5000];
    va_list ap;
    static int log_fd = -1;

    if (log_fd < 0) {
        char log_file_name[100];  /* Name of the log file */

        sprintf(log_file_name, "debug_log.%d", getpid());
        log_fd = open(log_file_name, O_WRONLY|O_CREAT, 0666);
    }
    va_start(ap, fmt);
    vsnprintf(full_string, sizeof(full_string), fmt, ap);
    if (log_fd >= 0) {
        write(log_fd, full_string, strlen(full_string));
    }
}
```

Remove the Muck

1. Go through the code and learn what it does.
2. Write down what you know as comments.
3. Remove dead code
4. Package code into simple, well understood, well tested modules.
5. Move common code to libraries

Muck Removal Demonstration

Paranoid Programming

or

Just because your paranoid, it doesn't mean they aren't out to get you.

Trust No One

- Validate all parameters
- All array accesses must be protected by an assert statement.

```
assert((index > 0) &&
       (index < N_ELEMENTS);
i = array[index];
```

The "debug_me" module

- Provides a "`debug_me`" function. Allows for "breakpoints" when you're not running a debugger.
- Catches failed asserts and starts the debugger.

Practice Safe Coding

- Risky

```
memcpy(buff1, buff2, BUFFER_SIZE);
memset(data_ptr, '\0',
  sizeof(struct data));
```

- Safe (r?)

```
memcpy(buff1, buff2,
  sizeof(buff1));
memset(data_ptr, '\0',
  sizeof(data_ptr[0]));
```

Practice Safe Coding

- Risky

```
memcpy(struct_ptr1, struct_ptr2,
  sizeof(struct the structure));
```

Safe (r?)

```
assert(struct_ptr1 != NULL);
assert(struct_ptr2 != NULL);
memcpy(struct_ptr1, struct_ptr2,
  sizeof(struct_ptr1[0]));
```

Practice Safe Coding

- Risky

```
strcpy(dest, src);
```

Better (actually less worse)

```
strncpy(dest, src, DEST_SIZE);
```

Safe

```
assert(dest != NULL);
assert(src != NULL);
strncpy(dest, src, sizeof(dest));
```

Practice Safe Coding

- Risky

```
strcat(dest, src);
```

- Better (actually less worse)

```
strncat(dest, src,
        DEST_SIZE-strlen(dest)-1);
dest[sizeof(dest)-1] = '\0';
```

Practice Safe Coding

- Safe

```
assert(dest != NULL);
assert(src != NULL);
strncat(dest, src,
    sizeof(dest)-strlen(dest)-1);
dest[sizeof(dest)-1] = '\0';
```

Practice Safe Coding

- Unsafe

```
gets(answer);
```

- Unsafe

```
strcpy(dest, src);
```

- Unsafe

```
memset(buffer, '\0', BUF_SIZE);
```

List of Dangerous Functions

Very Dangerous

- strcpy
- strcat
- gets

Moderately Dangerous

- memcpy
- memset
- malloc / free
- new / delete

Safe Free

- Bad

```
free(ptr);
```

- **Good**

```
assert(ptr != NULL);
free(ptr);
ptr = NULL; // Make sure it's
// not used again accidently
```

Safe Delete

- Always follow **`delete`** with the zeroing of the pointer

```
delete obj_ptr;
obj_ptr = NULL;
```

A Final Word

- Leave the code better than you find it.

A Final Word

- Leave the code better than you find it.
- It's usually not that hard!

Tools and References

glimpse	http://webglimpse.net
indent	http://www.gnu.org
Vim	http://www.vim.org
lxr	http://lxr.linux.no
ox	http://www.oualline.com
source navigator	http://sourcenav.sourceforge.net